NASA Technical Memorandum 88280

pEst Version 2.1 User's Manual

James E. Murray and Richard E. Maine

September 1987

National Aeronautics and
Space Administration

NASA Technical Memorandum 88280

pEst Version 2.1 User's Manual

James E. Murray and Richard E. Maine
Ames Research Center, Dryden Flight Research Facility, Edwards, California

1987

National Aeronautics and
Space Administration
Ames Research Center
Dryden Flight Research Facility
Edwards, California 93523-5000

CONTENTS

SUMMARY

This report is a user's manual for version 2.1 of pEst, a FORTRAN 77 computer program for interactive parameter estimation in nonlinear dynamic systems. The pEst program allows the user complete generality in defining the nonlinear equations of motion used in the analysis. The equations of motion are specified by a set of FORTRAN subroutines; a set of routines for a general aircraft model is supplied with the program and is described in the report. The report also briefly discusses the scope of the parameter estimation problem the program addresses. The report gives detailed explanations of the purpose and usage of all available program commands and a description of the computational algorithms used in the program.

INTRODUCTION

Parameter estimation techniques, in one form or another, have been in use at NASA Ames Research Center's Dryden Flight Research Facility (Ames-Dryden) and other research organizations for many years. High-speed digital computers were first used for parameter estimation in 1968 (ref. 1), and the number of parameter estimation computer programs available has since greatly increased. The MMLE3 program (modified maximum likelihood estimation program, version 3) developed at Ames-Dryden (ref. 2) has been accepted as an industry standard for aircraft parameter estimation and has been used on a variety of aircraft programs. The MMLE3 program is representative in two respects of the majority of parameter estimation programs currently in use. First, it is designed for use solely in a batch processing environment. Second, the equations of motion defining the dynamic model used in the program are linear. For a large class of well-behaved parameter estimation problems, these two characteristics pose no serious limitations in the utility of the program.

Recent flight test experience at Ames-Dryden has pointed out some of the limitations inherent in current parameter estimation programs. The dynamic behavior of aircraft at the extreme flight conditions currently being explored often cannot be appropriately modeled using the simple linear dynamic equations of motion. More accurate and flexible nonlinear models are often needed. The difficulties associated with extreme flight conditions, as well as those associated with the unique aircraft configurations currently being flown, have required significantly more attention from the analyst than previously. Interaction between the analyst and the estimation program is often the only viable means for obtaining results in a finite amount of time.

In response to these problems, Ames-Dryden researchers have developed a new parameter estimation program, pEst. The pEst program is designed to be fully interactive; however, it can be run in a batch mode. The program supports full nonlinear capability in the dynamic equations of motion; linear equations are acceptable as a subset.

This report documents the design philosophy, capabilities, and operational use of the pEst program. Section 1 defines the parameter estimation problem that pEst solves. Section 2 describes the philosophy of interactive program design as implemented in pEst. Section 3 describes the external files used by the program and how the program interfaces with other programs. Section 4 gives a description of each command in the pEst command set. Section 5 discusses various algorithms available during program use. Section 6 defines the standard user routines supplied with the program. Both the equations of motion and the definition of all system variables used in the equations are included. In this manual, file names, program prompts, and literal program input are shown in italics to distinguish them from other text. The appendixes contain information on file formats used by the program (app. A) and listings of the help files used in the program (app. B).

1 THE PARAMETER ESTIMATION PROBLEM

Conceptually, the parameter estimation problem is straightforward: We are studying a physical system, and we write a vector set of dynamic equations of motion that (hopefully) describes a model of the actual system. We presume to know the form of the equations but not the values of certain parametric variables in the equations. We perform an experiment with the actual physical system, recording the input to the sytem and the response of the system to the input. We seek to infer the values of the unknown parameters by adjusting their values in the model until its response agrees with the measured response.

The pEst program does two things. First, the program defines quantitatively the criterion for measuring the agreement between the model's computed response and the measured response. Second, it mechanizes the search for the unknown parameter values.

Figure 1 illustrates the pEst parameter estimation process. The number in each block refers to the section in this manual describing the function of the block.

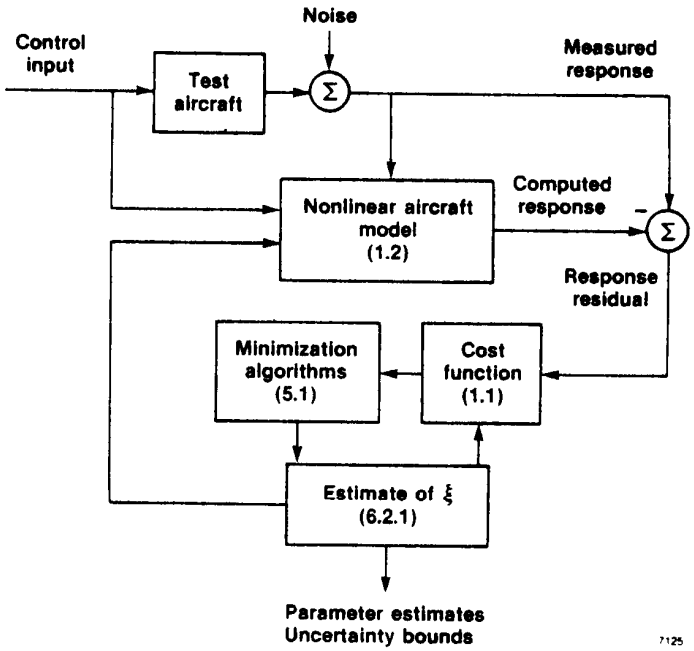

Figure 1. The pEst parameter estimation process.

1.1 Cost Function

The criterion is a scalar cost function that is an explicit function of the computed response and thus an implicit function of the vector of unknown parameters. The cost function used in the program is

$$J(\xi) = \frac{1}{2n_z n_t} \sum_{i=1}^{n_t} [z(t_i) - \tilde{z}(t_i)]^* W [z(t_i) - \tilde{z}(t_i)] \qquad (1)$$

where n_t and n_z are the numbers of time history points and response variables respectively, t is the time variable, W the response weighting matrix, z the measured response, $\tilde{z}$ the response computed by integrating the equations of motion, ξ the parameter vector, and superscript $*$ denotes transpose.

The cost function is quadratic in the computed response $\tilde{z}$.

1.2 Equations of Motion

The pEst program solves a vector set of time-varying, finite-dimensional, ordinary differential equations of motion. The equations are separated into the continuous-time state equation and the discrete-time response equation:

$$\dot{x}(t) = f[x(t), u(t), \xi] \quad (2)$$

$$z(t_i) = g[x(t_i), u(t_i), \xi] \quad (3)$$

where f is the state derivative function, g the response function, u the control variable, and x the state variable.

We have implemented a discrete-time feedback feature in the equations of motion. The feedback feature is similar in implementation and function to the process noise feature of the MMLE3 program (ref. 2). The nonlinear equations used in pEst, however, preclude using the discrete-time Kalman filter formulation of MMLE3; an ad hoc and intuitive approach is used in pEst. The feedback term is proportional to the difference between the measured and computed responses and is applied at each time point. The feedback gains k are parameters adjustable by the user (see section 6.2.1.4).

$$\hat{x}(t_i) = \tilde{x}(t_i) + k[z(t_i) - \tilde{z}(t_i)] \quad (4)$$

where $\hat{x}$ is the corrected estimate of the state variable and $\tilde{x}$ is the predicted estimate.

Input u and time t are assumed to be known exactly. The responses are measured at every sample point. There is no restriction that the sample rate be constant. The state derivative function f and the response function g are nonlinear functions of the parameter, state, input, and time. The specific form of the functions f and g is defined by a set of user-modifiable subroutines in pEst; the equations supplied with the program are documented in section 6.1.

2 INTERACTIVE DESIGN PHILOSOPHY IMPLEMENTED IN pEst

The first priority of any interactive program is a simple and efficient interface with the user; we have paid very close attention to the program's interface. Initially, we used a hierarchical menu-driven interface. However, we soon found the menu structure cumbersome, difficult, and sometimes even dangerous. We have rewritten the interface completely, adopting a simple command-driven interface. All program commands and capabilities are available for use at any time. Each command starts with a simple English key word; we also allow a wide range of synonyms and abbreviations for each command key word.

Error detection and correction are integral parts of any program, interactive or otherwise. A simple and responsive interface, however, gives the user greatly increased opportunities for making mistakes. In the interactive environment, good error handling becomes increasingly important. We have implemented error detection at every potential error source identified, and where possible we have applied error recovery procedures. We have made every attempt to make it impossible for you to crash the program. The program gives a one-line response to each error detected; we have attempted to give brief yet meaningful error messages. We leave all detailed explanations to the help files.

Online help files are also an integral part of any interactive program. Interactive program input calls for interactive troubleshooting of input errors. We have incorporated an online help facility into the

program. Each program command has its own help file detailing the use and syntax of the command. Additionally, we have incorporated information on subjects of interest to the program user into the help file system. All help files are accessible during program use; they are also available outside the program.

Interactive programs are generally reserved for difficult problems, where the approach to the solution is not clear at the outset. The solution progress tends to be discontinuous and incremental, with numerous dead ends met during the process. Efficient interactive problem solution requires a means of recovery from such dead ends. We have implemented a program feature that greatly enhances the potential to recover from errors and to restart the program if necessary. We have defined a program status file on which you can record the current status of the program at any point, thus allowing you to maintain an ancestor that can be used in the event of reaching a dead end. Effective use of the status file gives you freedom to experiment with different approaches to the problem, without fear of losing any progress already made.

The pEst program is an interactive program. Some problems do not require much user interaction to obtain a solution. If you can define a sequence of pEst commands that, when executed, will solve the problem, you can use the program in a batch operating mode.

3 INTERFACE TO OTHER PROGRAMS

The pEst package must be installed on your computer. Depending on the operating system and specific installation, a few features of the program may not be available (notably the help facility).

The pEst package consists of three separate programs. The pEst program itself is the parameter estimation program. The thPlot program plots time history data and time history fits. The GetData program (ref. 3) selects signals and maneuver times for analysis. In use, the pEst program interfaces with these and other programs through several files external to the program. All file names used by pEst and other programs are italicized in the following sections only to distinguish them from other text.

3.1 Measured Time History Data

A file of measured time history data for the case to be analyzed must be available in a form suitable to the program. The program reads the entire measured time history file into memory; the upper limit on the number of time points that the program can handle is 2000. This limit can be easily changed by a single-line modification in the program code. While you can interactively select subsets of the time history for use in the analysis (see section 4.7.8), it is most convenient for the measured time history to approximate the time interval or intervals to be used. The time between sample time points on the file need not be constant. The measured time history data file is never rewritten or altered by pEst.

Operationally, the measured time history file is divided into one or more time intervals called maneuvers. Each maneuver is treated as a complete and separate time history record in the integration of the equations of motion; the integration is reinitialized at the beginning of each maneuver. All maneuvers are used together in the estimation process; program variables, including estimated parameters, apply to all time points in all maneuvers. The program automatically breaks the time history into maneuvers when reading the file; a new maneuver is defined when one of two conditions is found. Both conditions are based on time values for successive time points read. If the time is nonincreasing or if the time increment exceeds a certain value (1.0 sec), then a new maneuver is started. In the simplest

and most common case, the file consists of a single maneuver. The maneuver times defined by the program may be displayed (see section 4.7.8).

By default, pEst expects the time history data to be on a file named *measured.* Other file names can be specified by user command.

The measured time history file is normally produced by the GetData program, which selects the desired times and signals from the available data. The pEst program recognizes signals by their names; therefore the signal names on the file must match those expected by pEst. The GetData program can rename signals if required. The signal names expected by the standard user routines are documented in section 6.2. The GetData program and the file format are documented separately in reference 3.

3.2 Program Status File

The program uses a status file to store the operational status of the program. With the exception of time history data, the status file stores the value of every program variable. Effective use of the status file is central to efficient use of the program. The status file serves three purposes. First, it can provide initial values for program variables and options at program startup. Second, it can store the program status to be used for later program recovery or restart. Third, it can store summary results at the conclusion of a run.

Efficient production use of pEst requires a status file at program startup. This file is not strictly required, as all variables are initialized with default values, which you can then change interactively. However, manually setting the many variables for each case is laborious if many cases are to be analyzed.

The initial status file can be obtained from one of several sources. For a large project, you will normally want to write a program that automatically creates an initial status file for each case; this program should get starting parameter estimates from the simulation data base. A status file (from whatever source) for one case can be copied and used to initialize another case; any required modifications can be made interactively. Making the required modifications is likely to take less effort than starting from scratch.

Program status can be saved or recovered at any time during program use. This feature provides a strong error-recovery capability. By saving the program status at appropriate intervals, you can continuously maintain a fallback position. Should you reach a dead end in the analysis, you can simply recover your previous program status and try a different approach to the problem.

Normal termination of pEst saves a file that reflects the complete program status prior to termination. This file can be used by any program that analyzes or displays the results of pEst. It can also be used with or without modifications as an initial status file for later cases.

By default, the program expects the file to be named *current*; other file names can be specified by user command.

The format of the status file is documented in appendix A.

3.3 Computed Time History Data

A file of time history data computed by the program is available for use outside the pEst program. The computed time history file is used by the thPlot program when plotting time history fits. It can also be used by several other programs.

The format of the computed time history file is identical to that of the measured time history file. The default name for the file is *computed*; other file names can be specified by user command.

3.4 Command Files

The pEst program is an interactive program, with commands normally entered singly from the terminal keyboard. Sometimes, however, you might have a sequence of commands that you will be executing as a group more than once. The program provides a way of automating such repetitive tasks. If you write the desired sequence of commands on a file, you can then instruct pEst to execute the commands from that file as though they were typed from the keyboard (see section 4.8.1). After executing all the commands in the file, control is returned to the terminal.

There is no default name for the command file; any name can be specified by user command.

3.5 Plot Commands File

The pEst program runs the thPlot program when plotting time history fits. A temporary scratch file is created to communicate information from pEst to thPlot whenever plotting. In normal operational use, this file is deleted after the successful completion of the plotting, so you should never be aware of its existence. In the event of catastrophic program failure, however, it is possible that this file, named *pEst_thPlot.temp*, might remain in existence.

4 HOW TO RUN THE PROGRAM

The pEst user interface is command driven; there is no menu. There is a single level of interaction between you and the program; all commands are available at any time. The program will accept commands when it issues the prompt *pest command*. The program recognizes a large set of commands; it is your responsibility to know the available command set and enter an appropriate command. Each command starts with a simple English key word and is followed by optional specifications. An interactive help facility is available to help you find both the appropriate command and the proper syntax of a command.

Each command controls a different aspect of the program; the applicable set of specifiers and their syntax are command dependent. Several specifiers, however, are used in the same syntax in several commands; their usage and syntax are documented in the following sections.

Several commands use switches to specify program options. A switch has one of two values; it turns an option either on or off. Most switches have one or more synonyms and antonyms, which are documented in the help files. Which name you use is a matter of personal preference and context; some fit more naturally into a certain command context than others. An algebraic sign is part of the switch specification; inverting the sign inverts the value of the switch. For example, −false is equivalent to +true.

Several commands also use key word and value pairs to specify program options. A key word and value pair assigns a value to a program variable. The key word is a word or abbreviation recognized by the program that corresponds to a variable in the program. The key word is delimited by a space or an equals sign and is followed by a value of the correct type.

All input to pEst is case insensitive; you can use any mixture of upper- and lower-case letters. Most key words, program variable names and values, and switches can be abbreviated. The minimum

recognizable abbreviations are indicated by the underlined letters of each key word, variable, or switch as first described in the following sections. Many key words also have synonyms.

The following sections document the individual commands of the pEst command set.

4.1 Help Command

The *help* command is the most important command in the program; the online help facility provides comprehensive and detailed descriptions of program commands, variables, and subjects of interest. There are three classes of information available: commands, variables, and topics. The commands *help commands*, *help variables*, and *help topics* give you listings and short (one-line) descriptions of all available commands, program variables, and topics, respectively. If you use *help* with the name of a specific command, program variable, or topic, you get a detailed help file listing giving information on syntax and usage and a few examples. When using the help command with a specific command or variable name, you must use the full primary name of the command or variable; synonyms and abbreviations are not acceptable. If you use the help command with no arguments, you get a brief description of the help facility.

All help file listings are included in appendix B.

The help command uses operating-system-specific software. Its implementation may differ depending on the operating system, and it may not be implementable on some systems.

Examples of the help command are

```
HELP
Help variables
help iterate
HELP PARAMETERS
```

4.2 Program Startup Commands

The program expects two files to be available at startup: a measured time history data file and an initial status file. If they are available, pEst automatically reads the measured time history file from the file *measured* and the initial program status from the file *current*. Both are default file names for the respective files. Two commands are available to get startup data from different files or to restart without exiting the program.

4.2.1 Read command.—Use the *read* command to read a measured time history data file into program memory. At this time, the program checks to see if all signals defined in the program (see section 6.2) are found on the file. If a signal is not found on the file, a constant value of zero is used for the time history of that signal. Some program variables, such as maneuvers (see section 3.1) and windows (see section 4.7.8), are automatically reset whenever a read command is executed. You can specify the name of the file to be read; if you omit the file name, the program uses the name last specified in a read command. The program automatically attempts to read the file *measured* at startup.

If the file does not exist or the program cannot successfully read the file for any reason, no time history data are stored in memory, an error message is printed, and control is returned to the command line.

If there are no measured time history data in program memory, some commands will not be accepted by the program. In particular, any command requiring integration of the equations of motion (such as iterating) will not be accepted; if you attempt any such command, an error message is printed, and control is returned to the command line.

Examples of read commands are

```
READ
Read measured.case6
```

4.2.2 Restore command.—Use the *restore* command to read the program status from a status file. You can specify the name of the file to be read; if you omit the file name, the program uses the name last specified in a restore command. The program automatically attempts to read the file *current* at startup.

If the file does not exist when you attempt to read it, an error message is printed, and control is returned to the command line. If the program fails when reading a file, all program variables successfully read into program memory up to that point are retained, an error message is printed, and control is returned to the command line.

Examples of restore commands are

```
Restore
REST  current.flt18.man6
```

4.3 Program Termination Commands

Several commands are available for terminating the program. Program status can be saved at program termination if desired.

4.3.1 Save command.—Use the *save* command to write the program status to a status file. You can specify the name of the file to be written; if you omit the file name, the program uses the name last specified in a restore command. The default file name at program startup is *current*. If a file with the specified name exists, it is overwritten.

If the program fails when writing a status file, no status file is written, an error message is printed, and control is returned to the command line.

Examples of save commands are

```
Save
save  curr.final
```

4.3.2 Quit command.—Use the *quit* command to write the current program status to a status file and then terminate the program. This is exactly equivalent to running, in order, the save and abort commands. Note that you cannot specify the name of the status file to be written; the program uses the name last specificed in a restore command.

If the program fails when writing the status file, the file is not written, an error message is printed, and the program terminates.

An example of the quit command is

```
quit
```

4.3.3 Abort command.—Use the *abort* command to terminate the program and return control to the operating system. This command does not save the current program status; any progress made since you last ran a save command is lost.

An example of the abort command is

```
Abort
```

4.4 Plotting Commands

The pEst program uses the thPlot program for plotting time histories of response variable fits and other variables. The pEst program communicates with thPlot through a file containing computed time history data.

The equations of motion are integrated to produce the time history of the computed variables. All computed variables are first assigned constant default values for the entire time history. The program then integrates the equations of motion using current program status, replacing the default values with the computed values as the integration proceeds. If the integration fails for any reason, the computed time histories up to the point of failure are stored, an error message is printed, and the integration is terminated.

4.4.1 Write command.—Use the *write* command to write the computed time history data from program memory to an external file. Time histories for all computed state and response variables are written to the file. The time histories of all computed variables are made consistent with the current program status prior to writing the computed time history file by integrating the equations of motion using the current program status. Time histories for all response residual variables, as well as measured control, response, and extra variables, are also written to the file. You can specify the name of the file to be written; if you omit the file name, the program uses the name last specified in a write command. The default file name at program startup is *computed.*

If a file with the specified name already exists, it is overwritten. If no measured time history data are in program memory, the computed time history file is not written, an error message is printed, and control is returned to the command line. If the integration terminates prematurely for any reason, an error message is printed, but the computed time history file is still written. If the program fails to write the file, an error message is printed, and control is returned to the command line.

Examples of the write command are

```
write
Write computed.maneuver10g
```

4.4.2 Plot command.—Use the *plot* command to plot time history fits for all active response variables (see section 4.6.4) using the thPlot program. The computed time histories are first written to an external file by automatically running a write command. The time histories are plotted four per

page; multiple pages are plotted if required. The measured response and computed response are both plotted on the same axis; the axis is automatically scaled to accommodate both variables. You can optionally request plots of additional time history variables (response residuals, states, controls, and extras); the plots for these variables follow the plots of the time history fits of the response variables.

If no measured time history data exist in program memory, an error message is printed, and no plots are made. If the integration terminates prematurely for any reason, an error message is printed, but plots are still made. If the program fails to write the computed history file for any reason, an error message is printed, and no plots are made.

Examples of the plot command are

```
Plot
plot  +resids  +states
```

4.4.3 thPlot command.—Use the *thPlot* command to run the thPlot program from within pEst. The computed time histories are first written to an external file by automatically executing the write command. The pEst program does not communicate with thPlot when using this command; you have complete freedom to read any files and plot any signals desired. After thPlot terminates, control is returned to pEst.

An example of the thPlot command is

```
thplot
```

4.5 Iterate Command

Use the *iterate* command to start the iterative parameter estimation process. You can specify the maximum number of iterations and the desired minimization algorithm. If you do not specify the number of iterations, no iterations are done; the equations of motion are integrated, and the cost function is evaluated. The three minimization algorithms available are gradient, Newton-Raphson, and Davidon-Fletcher-Powell. If you omit the minimization algorithm, the algorithm last specified remains in effect. The default algorithm at program startup is Newton-Raphson. All the iterations specified with a single iterate command use the same minimization algorithm.

Before attempting to iterate, the program tests the validity of the current program status for estimation; if the validity test fails, iterating is not allowed, and control is returned to the command line. For example, if no measured time history data are in program memory, iterating will not be allowed. If all validity tests succeed, the program iterates until either the specified number of iterations is completed or the convergence criterion (see section 4.7.4) is met. The iterative process may terminate with one or several error messages if any of numerous problems is detected. If the estimation is prematurely terminated, all progress made up to the point of failure is retained and is reflected in the current program status.

Examples of the iterate command are

```
Iterate  6 newton-raphson
it 3
IT
```

4.6 System Variable Commands

The equations of motion define the dynamic system analyzed by the program. The equations of motion contain numerous time history and parametric variables defining the specific characteristics of the dynamic system. The system variable commands allow you to display and to modify these variables and therefore the characteristics of the dynamic system. The time history and parametric variables are vector variables, and each vector element has several characteristics or attributes. Each system variable command allows you to display and to modify the attributes of selected elements of a specified vector.

The first part of each system variable command selects elements of the vector to be displayed or modified. The syntax of this specification is common to all system variable commands. A generalized list of vector elements follows the command name. The list can be a list of element names, delimited by commas or blanks. It can also be one of two key words, *all* or *active*; *all* selects all elements of the vector, and *active* selects the vector elements with active status. The definition of active status is dependent on the vector and is defined in each of the following command descriptions. If the list is omitted, all active elements are selected by default. Whenever you execute a system variable command, the values of the selected elements are displayed.

For each system variable command, several optional descriptors control the attributes to be displayed or modified. The order in which they are specified on the command line is immaterial. The descriptors and the syntax for using them are documented in each of the following command descriptions.

4.6.1 Parameter command.—Use the *parameter* command to display and modify the attributes of the parameters. The parameters are the variables that can be estimated by the program. Each parameter has five attributes: current value, estimation status, predicted value, Cramér-Rao bound, and change in value from previous iteration. The current values of the selected parameters are always displayed after the command is entered. The estimation status of a parameter is active or inactive; an active parameter is one that is currently being estimated, while an inactive parameter is one that is not. You can modify the estimation status of the selected parameters with the $\pm$*active* switch. The predicted value of a parameter is fixed and can only be displayed with the $\pm$*predicted* switch. The Cramér-Rao bound and the change in parameter value from the previous iteration are defined in the estimation process and cannot be directly modified by the user; they can only be displayed with the $\pm$*bound* and $\pm$*delta* switches, respectively. You can modify the current value of a parameter in one of two ways. You can explicitly specify a value, and that value is assigned to all selected parameters. Alternatively, you can turn on the $\pm$*restore* switch, and each selected parameter will be reset to its corresponding predicted value.

The parameters defined by the standard user routines are documented in section 6.2.1.

Examples of parameter commands are

```
PARAMETER  clp=-0.25
par  cma,cNorma cmde,cNormde  +on
Par Active +Restore
parm +cr +delta
```

4.6.2 Constant command.—Use the *constant* command to display and modify the constants in the program. The constants are program variables that cannot be estimated by the program. The

value of each selected constant is always displayed after the command is entered. You can modify the value of a constant; the specified value is assigned to all selected constants.

The constants defined by the standard user routines are documented in section 6.2.2.

Examples of constant commands are

```
constant  mass=1056.0
CONST ixy,iyz 0.
Const all
```

4.6.3 State command.—Use the *state* command to display and modify the attributes of the state variables in the program. A state variable has two attributes: its status and its integration limit. If a state is active, the state equation for the state is integrated, and this integrated value is used in the equations of motion. If a state is inactive, the state equation is not integrated, and the equations of motion are modified to remove the corresponding state equation. You can modify the status of the selected state variables with the *+active* switch. The integration limit for a state variable is used to avoid catastrophic program failure during integration of the equations of motion. If during integration the value of a state variable exceeds its limit, an error message is printed, and the integration is terminated. The integration limit is a floating-point value specified using the *limit* key word and value pair.

The state variables defined by the standard user routines are documented in section 6.2.3.

Examples of state commands are

```
state  v,alpha,an,q  +on
STATE p r  lim=10000.
```

4.6.4 Response command.—Use the *response* command to display and modify the attributes of the response variables in the program. A response variable has two attributes: its status and its weighting. The status of a response variable determines whether or not the response variable time history is computed. If a response is active, the response variable time history is computed and made available for other uses. If a response is inactive, no response time history is computed. You can modify the status of the selected response variables with the *+active* switch. The weighting of a response variable specifies the variable's weighting in the cost function and is specified with the *weight* key word and value pair. The key word *output* is a synonym for response.

Note that making a response variable inactive is not equivalent to making its weighting zero. If a response variable is made inactive, the equations of motion are modified to remove the corresponding response equation. There may also be secondary changes in the equations of motion to remove all usage of the response variable. These secondary changes depend on the equations of motion used. If the response is active but has zero weighting, the response is computed and used in the equations of motion but does not directly influence the cost function.

The response variables defined by the standard user routines are documented in section 6.2.5.

Examples of response commands are

```
response  beta p r  +on
OUT alpha  w=150.
```

4.6.5 Flag command.—Use the *flag* command to display and modify the flags in the program. A flag is a logical variable used in the equations of motion. The flags typically select alternative forms of the equations of motion or sources of data. You can modify the value of a flag with the *+on* switch; the switch value is assigned to all selected flags.

The flags defined by the standard user routines are documented in section 6.2.7.

Examples of flag commands are

```
flag use_avg_qbar
Flag  use_avg_alpha,use_avg_beta  +ON
FLAG  all +off
```

4.7 Program Variable Commands

Use the *set* and *show* commands to display and modify the program variables and options controlling the estimation process.

The set command sets the value of program variables. With one exception, the command syntax is the command key word followed by a key word and value pair specifying a program variable and setting its value. The valid key words and the program features they control are described in the following sections. Each key word is given with its first few letters underlined; the underlined portion is the minimum abbreviation recognized by the program. The value type is dependent on the variable; for each variable described, the range of legal values is specified. Any variable modified by the command is displayed.

The show command allows you to display the values of program variables. The command syntax is the command key word followed by a variable name or appropriate abbreviation. In addition to the variables described by the set command, there is one variable that is only displayable.

4.7.1 Integration method variable.— The *integMeth* variable specifies the algorithm used when integrating the equations of motion. The variable has two possible values, *euler* and *runge-kutta*; the default value is *runge-kutta*.

Examples of the use of the integration method variable are

```
show  integMeth
SET integ runge
```

4.7.2 Gradient method variable.—The *gradMeth* variable specifies the algorithm used when computing the finite-difference gradients. The variable has two possible values: *single-sided* specifies the forward difference algorithm, and *double-sided* specifies the central difference algorithm. The default value is single-sided.

Examples of the use of the gradient method variable are

```
sh gradmethod
Set  GradM  2
```

4.7.3 Gradient delta variable.—The *gradDelta* variable specifies the parameter increment used in computing the finite-difference gradients. The parameter increment used for each parameter is defined by

$$d\xi = \text{gradDelta} * \max(\xi, 0.000001)$$

where ξ is the parameter value and $d\xi$ is the parameter increment.

Valid values are floating-point numbers; the program default value is 0.0000001.

Examples of the use of the gradient delta variable are

```
sh graddelta
SET  GRADD=0.0001
```

4.7.4 Convergence bound variable.—The *bound* variable defines the convergence criterion for the estimation process. If the percentage change in cost between two successive iterations drops below the value of the bound variable, convergence is declared, and the iterative process is terminated. Valid values for this variable are floating-point numbers; the program default value is 0.0001.

Examples of the use of the convergence bound variable are

```
SHOW  BOUND
set bound 1.0e-6
```

4.7.5 Message level variable.—The *msgLevel* variable controls the amount of output that the program prints during use. Higher values produce larger amounts of output. Valid values for the variable are integers between 0 and 100; the default value is 50. The help file lists the significance of the various numerical values.

Examples of the use of the message level variable are

```
sho msg
Set  MsgLev=65
```

4.7.6 Plot title variable.—The *title* variable specifies the title used on the time history plots of response fits. Valid values are character strings of up to 40 characters. If there are blanks embedded in the string, you must put the whole string inside quotes. The default value for the title is blank.

Examples of the use of the plot title variable are

```
Show Tit
SET  title  'Space Shuttle Flight 4 Maneuver 3b'
```

4.7.7 Statistics variable.—The *statistics* variable contains sample statistics of all measured variables from the time history data file. The statistics are defined whenever a measured time history file is read and may not be modified during use; they may be displayed only with the show command.

An example of the use of the statistics variable is

```
Sho Stats
```

4.7.8 Maneuver window variable.—A window is a time subset of the measured time history file. Each window must be wholly contained within a maneuver (see section 3.1). When the equations of motion are integrated, the integration is reinitialized at the start of each window. Only the time points within the windows are used in the analysis. Integration of the equations of motion defines values for all computed variables for all time points in all maneuvers, regardless of whether the time points are inside or outside the windows. For time points outside the windows, the program uses a constant value for each computed variable; the value depends on the variable type. For a state variable, the value is zero; for a response variable, the value is the average value of the measured data for the variable. At program startup, the default window or windows are identical to the maneuver or maneuvers.

The *window* variable specifies the window or windows used in the analysis. The window variable specification has three elements: the window number, the maneuver number, and the time specification. Whenever a window is referenced, all currently defined maneuvers and windows are displayed.

Examples of the use of the window variable are

```
Show windows
SET  WINDOW  TIME  0 - 10
set  wind 2  man 2  time 0.5 7.5
```

4.8 Advanced Commands

4.8.1 Do command.—Use the *do* command to read a sequence of pEst commands from an external command file and execute them. Upon successful completion of the command sequence, control is returned to the command line.

Examples of the do command are

```
do initialize
Do Startup.X29
```

4.8.2 System command.—Use the *system* command to execute an operating system command from within pEst. This command may not be implemented on some systems.

Examples of system commands are

```
SYSTEM FILES
sys help copy
sys Rename Current.init.f18 Current
```

5 ALGORITHMS

The pEst program gives you selective use of several different algorithms for various program functions. The algorithms currently available are described in the following sections.

5.1 Minimization Algorithms

The pEst program has three algorithms available for iteratively minimizing the cost function: gradient (or steepest descent), modified Newton-Raphson, and Davidon-Fletcher-Powell. The Newton-Raphson

algorithm is modified to eliminate an undesirable characteristic it has when used far from the minimum of the cost function (or in any other case where the cost function is far from quadratic in the parameter vector). It is not uncommon in such a case for a strict Newton-Raphson iteration to produce a higher cost value. To alleviate this problem, the Newton-Raphson algorithm is implemented with an explicit line search; first, an initial parameter increment is computed using the Gauss-Newton algorithm; then the parameter space is searched along the line defined by the parameter increment for the minimizing cost value. The addition of the line search means that a Newton-Raphson iteration is guaranteed to produce a cost value no larger than the cost before the iteration. Both the gradient and Davidon-Fletcher-Powell algorithms use implicit line searches; an iteration using either algorithm is also guaranteed to produce a nonincreasing cost value.

Any or all of the available minimization algorithms may be used on a single problem. The utility of any algorithm depends on both the problem and the location in the parameter space with respect to the minimum. It is commonly useful to start the minimization with one algorithm and to later switch to a different algorithm to complete the solution. The different algorithms do not interact.

The gradient algorithm uses only first-gradient information and consequently performs best where the cost function is very steep. It is useful for greatly reducing the cost when far from the minimum, as is common when beginning work on the problem. However, its performance deteriorates as it approaches the minimum, and the cost becomes flatter. In cases of high correlation between parameters, it may even stall so completely as to appear to have converged. We do not recommend using the gradient algorithm for the final iteration.

The Newton-Raphson algorithm uses second-gradient information in addition to first gradient information and consequently performs best where the cost function is approximately quadratic. This approximation is generally most accurate near the minimum of the cost function; the algorithm has excellent convergence characteristics once it is close to the minimum. However, the Newton-Raphson algorithm is sensitive to identifiability problems; the algorithm may fail if there are linear dependencies among the parameters. The Newton-Raphson algorithm is the only algorithm that can compute Cramér-Rao bounds, which are defined for all active parameters following a successful Newton-Raphson iteration.

The Davidon-Fletcher-Powell algorithm, in some sense, combines the best features of the gradient and Newton-Raphson algorithms. This algorithm changes character from iteration to iteration. Initially, it performs much like the gradient algorithm; the first iteration is identical to a gradient iteration. As the iterations proceed, the algorithm gains information on the second gradient and approaches the Newton-Raphson algorithm in character. Thus, this algorithm combines the initially rapid cost reduction characteristics of the gradient algorithm with the excellent convergence characteristics of the Newton-Raphson algorithm.

The iterative process is automatically terminated if a convergence criterion is met. The convergence criterion is based on cost values for successive iterations; if the percentage change in the cost value drops below a threshhold value (see section 4.7.4), convergence is declared, and the iterative process is terminated.

5.2 Gradient Computation

The minimization algorithms require computation of first or second, or both, gradients of the cost function at each iteration. The complete generality in the nonlinear equations of motion precludes using analytical differentiation to compute the gradients; finite-difference algorithms are used. The

parameter increment used in computing the gradients is a fixed percentage of the parameter value (see section 4.7.3).

Both single-sided and central difference algorithms are supported. Gradient computation using the single-sided difference algorithm requires $n_p + 1$ solutions of the equations of motion, where n_p is the number of parameters; the central difference algorithm requires $2n_p$ solutions of the equations of motion. The gradient computation requires a large number of solutions of the equations of motion; the computational time involved is typically the most significant percentage of the total time used in an iteration. The single-sided difference algorithm is the faster of the two algorithms, but it is also the less accurate. The increased accuracy of the central difference algorithm, however, is not really consequential until close to convergence; we generally find it most efficient to use single-sided differences for all but the last few iterations. The Gauss-Newton approximation (ref. 2) is used to compute the second gradient, thus the computational burden of computing the second gradient in addition to the first gradient is not significant.

5.3 Integrating the Equations of Motion

There are two algorithms available for integrating the equations of motion: forward Euler and fourth-order Runge-Kutta integration. The forward Euler integration requires one evaluation of the state derivative function f for each time point of the solution, while the Runge-Kutta algorithm requires four such function evaluations per time point. Both algorithms require a single evaluation of the response function g for each time point of the solution. Thus, the Euler algorithm is the faster of the two. The Runge-Kutta algorithm, though slower, is more accurate and has a larger region of stability.

The nonlinear character of the functions f and g requires consideration of several issues that are not relevant for linear equations of motion. The inherent possibility of singularities in f and g means that catastrophic termination of the program (with subsequent loss of anything done up to that point) is a real consideration. Extreme caution (maybe even paranoia, as some have suggested) is necessary to anticipate and eliminate these possibilities. Singularities, however, are just extreme and obvious cases of ill conditioning; the more subtle cases can also generate equally catastrophic errors. In the estimation process, it is not uncommon for an intermediate solution to diverge greatly from the final converged solution. The large magnitudes of the state and observation variables in these intermediate solutions can easily exceed the bounds of validity of intrinsic FORTRAN functions. We have implemented limit checking on the state variables (see section 4.6.3) to preclude catastrophic computational errors.

6 STANDARD USER ROUTINES

The pEst program is supplied with a set of equations of motion to model a wide range of aircraft problems. The equations of motion are a full six-degree-of-freedom nonlinear set of differential equations. We do not assume that the aircraft is symmetric. We do assume fixed aircraft geometry and constant mass characteristics. No propulsion or rotating-mass effects are included. We assume a flat earth and constant gravitational acceleration. We use English units throughout the equations.

6.1 Equations of Motion

The standard user routines define the equations of motion used by the program. The equations are divided into the state equations and the response equations. All time history and parameteric variables

used in the equations are described in section 6.2. There is no explicit division of the equations into longitudinal and lateral-directional subsets. The state equation for airspeed is not implemented in pEst.

The standard user routines define 8 state equations, 5 feedback equations, and 14 response equations. Only in rare instances, however, will you use the complete set of equations. In practice you will probably interactively define a subset that is dependent on the maneuver being analyzed.

You can independently activate the state equations and response equations. Each state equation is integrated only if it is active (see section 4.6.3); inactive state equations are not integrated and are removed from the equations of motion. Each response equation is evaluated only if it is active (see section 4.6.4); inactive response equations are removed from the equations.

Each state variable on the right-hand side of the state and response equations can come from one of three sources. First, the computed time history of the state variable can be used, if available. Second, the corresponding measured response variable can be used, if available. Third, for some state variables, a constant value can be used. The program flags and the status of the state variables determine the source of each state variable on the right-hand side of the equations. Five flags specify using average values of measurement variables (V, α, β, θ, and ϕ) in the equations. If a flag is turned on, the value of the corresponding constant variable is used in the equations. If the flag is not turned on (or if there is no corresponding flag for the state variable), the status of the state variable determines the source. If the state variable is active, the computed state time history is used. If the state variable is inactive, the corresponding measurement variable is used.

Two additional flags specify the source of the extra time history variables (such as $\bar{q}$) on the right-hand side of the equations. If the flag for an extra variable is turned on, the value of the corresponding constant variable is used; otherwise the measured time history of the extra variable is used.

6.1.1 State equations.—The predicted state variables $\tilde{x}$ are obtained by integrating the state equations defined in the following equations.

$$
\begin{aligned}
\dot{\alpha} &= q - \tan\beta(p\cos\alpha + r\sin\alpha) - \frac{\bar{q}s\mathrm{R}}{mV\cos\beta}C_L \\
&\quad + \frac{g\mathrm{R}}{V\cos\beta}(\cos\theta\cos\phi\cos\alpha + \sin\theta\sin\alpha) \\
\dot{\beta} &= p\sin\alpha - r\cos\alpha + \frac{\bar{q}s\mathrm{R}}{mV}C_Y \\
&\quad + \frac{g\mathrm{R}}{V}[\cos\beta\cos\theta\sin\phi - \sin\beta(\cos\theta\cos\phi\sin\alpha - \sin\theta\cos\alpha)] \\
I_x\dot{p} - I_{xy}\dot{q} - I_{xz}\dot{r} &= \bar{q}sbC_\ell\mathrm{R} + [qr(I_y - I_z) + (q^2 - r^2)I_{yz} + pqI_{xz} - prI_{xy}]/\mathrm{R} \\
I_y\dot{q} - I_{yz}\dot{r} - I_{xy}\dot{p} &= \bar{q}scC_m\mathrm{R} + [pr(I_z - I_x) + (r^2 - p^2)I_{xz} + qrI_{xy} - pqI_{yz}]/\mathrm{R} \\
I_z\dot{r} - I_{xz}\dot{p} - I_{yz}\dot{q} &= \bar{q}sbC_n\mathrm{R} + [pq(I_x - I_y) + (p^2 - q^2)I_{xy} + prI_{yz} - qrI_{xz}]/\mathrm{R} \\
\dot{\theta} &= q\cos\phi - r\sin\phi \\
\dot{\phi} &= p + \tan\theta(r\cos\phi + q\sin\phi)
\end{aligned}
$$

where

b	is reference span,
c	reference chord,
C_L	coefficient of lift,

C_ℓ	coefficient of rolling moment,
C_m	coefficient of pitching moment,
C_n	coefficient of yawing moment,
C_Y	coefficient of lateral force,
g	gravitational acceleration,
I_x, I_y, I_z	are moments of inertia,
I_{xy}, I_{xz}, I_{yz}	cross products of inertia,
m	is mass,
p	roll rate,
q	pitch rate,
$\bar{q}$	dynamic pressure,
r	yaw rate,
R	conversion factor (57.2958),
s	reference area,
V	total velocity,
α	angle of attack,
β	angle of sideslip,
θ	pitch attitude, and
ϕ	roll attitude.

6.1.2 Initial conditions.—The initial conditions of the state variables used in integrating the equations are defined as follows.

$$
\begin{aligned}
V(0) &= V_{z_0} - V_{\mathrm{b}} + V_0 \\
\alpha(0) &= (\alpha_{z_0} - \alpha_{\mathrm{b}})/k_\alpha + \alpha_0 \\
\beta(0) &= (\beta_{z_0} - \beta_{\mathrm{b}})/k_\beta + \beta_0 \\
p(0) &= p_{z_0} - p_{\mathrm{b}} + p_0 \\
q(0) &= q_{z_0} - q_{\mathrm{b}} + q_0 \\
r(0) &= r_{z_0} - r_{\mathrm{b}} + r_0 \\
\theta(0) &= \theta_{z_0} - \theta_{\mathrm{b}} + \theta_0 \\
\phi(0) &= \phi_{z_0} - \phi_{\mathrm{b}} + \phi_0
\end{aligned}
$$

where $f(0)$ is the value of the state variable at the beginning of the integration, subscript b denotes the measurement bias for the corresponding observation variable, subscript 0 denotes the increment to the initial state value, and subscript z_0 denotes the measured value of the corresponding observation variable of the initial point.

6.1.3 Total force and moment coefficients.— The total force and moment coefficients used in the state and response equations are defined as follows.

$$
\begin{aligned}
C_L &= C_N \cos\alpha - C_A \sin\alpha \\
C_A &= C_{A_0} + C_{A_\alpha}\alpha + \frac{c}{2V\mathrm{R}} C_{A_q} q + C_{A_{\delta_e}}\delta_{\mathrm{e}} + C_{A_{\delta_1}}\delta_1 + C_{A_{\delta_2}}\delta_2 + C_{A_{\delta_3}}\delta_3 \\
C_N &= C_{N_0} + C_{N_\alpha}\alpha + \frac{c}{2V\mathrm{R}} C_{N_q} q + C_{N_{\delta_e}}\delta_{\mathrm{e}} + C_{N_{\delta_1}}\delta_1 + C_{N_{\delta_2}}\delta_2 + C_{N_{\delta_3}}\delta_3 \\
C_Y &= C_{Y_0} + C_{Y_\beta}\beta + \frac{b}{2V\mathrm{R}}(C_{Y_p} p + C_{Y_r} r) + C_{Y_{\delta_a}}\delta_{\mathrm{a}} + C_{Y_{\delta_r}}\delta_{\mathrm{r}} + C_{Y_{\delta_3}}\delta_3 + C_{Y_{\delta_4}}\delta_4
\end{aligned}
$$

$$
\begin{aligned}
C_\ell &= C_{\ell_0} + C_{\ell_\beta}\beta + \frac{b}{2V\mathrm{R}}(C_{\ell_p}p + C_{\ell_r}r) + C_{\ell_{\delta_\mathrm{a}}}\delta_\mathrm{a} + C_{\ell_{\delta_\mathrm{r}}}\delta_\mathrm{r} + C_{\ell_{\delta_3}}\delta_3 + C_{\ell_{\delta_4}}\delta_4 \\
C_m &= C_{m_0} + C_{m_\alpha}\alpha + \frac{c}{2V\mathrm{R}}C_{m_q}q + C_{m_{\delta_\mathrm{e}}}\delta_\mathrm{e} + C_{m_{\delta_1}}\delta_1 + C_{m_{\delta_2}}\delta_2 + C_{m_{\delta_3}}\delta_3 \\
C_n &= C_{n_0} + C_{n_\beta}\beta + \frac{b}{2V\mathrm{R}}(C_{n_p}p + C_{n_r}r) + C_{n_{\delta_\mathrm{a}}}\delta_\mathrm{a} + C_{n_{\delta_\mathrm{r}}}\delta_\mathrm{r} + C_{n_{\delta_3}}\delta_3 + C_{n_{\delta_4}}\delta_4
\end{aligned}
$$

where

C_A	is coefficient of axial force,
C_N	coefficient of normal force,
δ_a	aileron deflection,
δ_e	elevator deflection,
δ_r	rudder deflection,
δ_1	control surface 1 deflection,
δ_2	control surface 2 deflection,
δ_3	control surface 3 deflection,
δ_4	control surface 4 deflection,
$C_{A_0}, C_{A_\alpha}, C_{A_q}, C_{A_{\delta_\mathrm{e}}}, C_{A_{\delta_1}}, C_{A_{\delta_2}}, C_{A_{\delta_3}}$	are axial force parameters,
$C_{N_0}, C_{N_\alpha}, C_{N_q}, C_{N_{\delta_\mathrm{e}}}, C_{N_{\delta_1}}, C_{N_{\delta_2}}, C_{N_{\delta_3}}$	normal force parameters,
$C_{Y_0}, C_{Y_\beta}, C_{Y_p}, C_{Y_r}, C_{Y_{\delta_\mathrm{a}}}, C_{Y_{\delta_\mathrm{r}}}, C_{Y_{\delta_3}}, C_{Y_{\delta_4}}$	lateral force parameters,
$C_{\ell_0}, C_{\ell_\beta}, C_{\ell_p}, C_{\ell_r}, C_{\ell_{\delta_\mathrm{a}}}, C_{\ell_{\delta_\mathrm{r}}}, C_{\ell_{\delta_3}}, C_{\ell_{\delta_4}}$	rolling moment parameters,
$C_{m_0}, C_{m_\alpha}, C_{m_q}, C_{m_{\delta_\mathrm{e}}}, C_{m_{\delta_1}}, C_{m_{\delta_2}}, C_{m_{\delta_3}}$	pitching moment parameters, and
$C_{n_0}, C_{n_\beta}, C_{n_p}, C_{n_r}, C_{n_{\delta_\mathrm{a}}}, C_{n_{\delta_\mathrm{r}}}, C_{n_{\delta_3}}, C_{n_{\delta_4}}$	yawing moment parameters.

All parameters are defined in section 6.2.

6.1.4 Response equations.—The computed response variables $\tilde{z}$ are obtained by evaluating the response equations defined in the following equations.

$$
\begin{aligned}
V_{\tilde{z}} &= V + V_b \\
\alpha_{\tilde{z}} &= k_\alpha\left[\alpha + (x_\alpha - x_{\mathrm{cg}})\frac{q}{V} + (y_\alpha - y_{\mathrm{cg}})\frac{p}{V}\right] + \alpha_b \\
\beta_{\tilde{z}} &= k_\beta\left[\beta + (z_\beta - z_{\mathrm{cg}})\frac{p}{V} - (x_\beta - x_{\mathrm{cg}})\frac{r}{V}\right] + \beta_b \\
p_{\tilde{z}} &= p + p_b \\
q_{\tilde{z}} &= q + q_b \\
r_{\tilde{z}} &= r + r_b \\
\theta_{\tilde{z}} &= \theta + \theta_b \\
\phi_{\tilde{z}} &= \phi + \phi_b \\
a_{x_{\tilde{z}}} &= -\frac{\bar{q}s}{mg}C_A - \frac{1}{g\mathrm{R}}[(z_{a_x} - z_{\mathrm{cg}})\dot{q} + (y_{a_x} - y_{\mathrm{cg}})\dot{r}] + \frac{1}{g\mathrm{R}^2}(x_{a_x} - x_{\mathrm{cg}})(q^2 + r^2) + a_{x_b}
\end{aligned}
$$

$$
\begin{aligned}
a_{y_{\tilde{z}}} &= \frac{\bar{q}s}{mg}C_Y + \frac{1}{g\mathrm{R}}[-(x_{a_y} - x_{\mathrm{cg}})\dot{r} + (z_{a_y} - z_{\mathrm{cg}})\dot{p}] - \frac{1}{g\mathrm{R}^2}(y_{a_y} - y_{\mathrm{cg}})(p^2 + r^2) + a_{y_b} \\
a_{n_{\tilde{z}}} &= \frac{\bar{q}s}{mg}C_N - \frac{1}{g\mathrm{R}}[(x_{a_n} - x_{\mathrm{cg}})\dot{q} + (y_{a_n} - y_{\mathrm{cg}})\dot{p}] - \frac{1}{g\mathrm{R}^2}(z_{a_n} - z_{\mathrm{cg}})(q^2 + p^2) + a_{n_b} \\
\dot{p}_{\tilde{z}} &= \dot{p} + \dot{p}_b \\
\dot{q}_{\tilde{z}} &= \dot{q} + \dot{q}_b \\
\dot{r}_{\tilde{z}} &= \dot{r} + \dot{r}_b
\end{aligned}
$$

where

$\dot{p}$	is roll acceleration,
$\dot{q}$	pitch acceleration,
$\dot{r}$	yaw acceleration,
$x_{a_x}, y_{a_x}, z_{a_x}$	are axial accelerometer position parameters,
$x_{a_y}, y_{a_y}, z_{a_y}$	lateral accelerometer position parameters,
$x_{a_n}, y_{a_n}, z_{a_n}$	normal accelerometer position parameters,
$x_{\mathrm{cg}}, y_{\mathrm{cg}}, z_{\mathrm{cg}}$	aircraft center-of-gravity position constants,
$k_\alpha, x_\alpha, y_\alpha$	angle-of-attack measurement parameters,
$k_\beta, x_\beta, z_\beta$	angle-of-sideslip measurement parameters,

and the subscript $\tilde{z}$ denotes the computed response value.

6.1.5 State feedback equations.— The corrected state variables $\hat{x}$ are obtained by evaluating the discrete feedback equations (see section 1.2) defined by the following standard user routines. Feedback is implemented for only five of the state variables.

$$
\begin{aligned}
\hat{\alpha} &= \tilde{\alpha} + (g_\alpha/k_\alpha)(\alpha_{\tilde{z}} - \alpha_z) \\
\hat{\beta} &= \tilde{\beta} + (g_\beta/k_\beta)(\beta_{\tilde{z}} - \beta_z) \\
\hat{p} &= \tilde{p} + g_p(p_{\tilde{z}} - p_z) \\
\hat{q} &= \tilde{q} + g_q(q_{\tilde{z}} - q_z) \\
\hat{r} &= \tilde{r} + g_r(r_{\tilde{z}} - r_z)
\end{aligned}
$$

where $g_\alpha, g_\beta, g_p, g_q$, and g_r are feedback gain parameters, the superscript $\tilde{}$ denotes the predicted state value, and the superscript $\hat{}$ denotes the corrected state value.

6.2 System Variables and Names

The standard user routines define the names of all system variables: parameters, constants, states, controls, responses, extras, and flags. In program use, all system variables are accessed by their given names.

6.2.1 Parameters.—The standard user routines define 97 parameters. Parameter subsets are defined below. Each parameter is individually documented in the parameters help file.

6.2.1.1 Stability and control derivatives: Any parameter whose name starts with the letter c is a stability or control derivative. There are three classes of stability and control derivatives: aerody-

namic biases, control derivatives, and stability derivatives. Each derivative is nondimensionalized; the details of the nondimensionalization depend on the parameter. All aerodynamic biases are completely normalized; they are dimensionless. All rotational rate derivatives are expressed in reciprocal radians. All α, β, and control derivatives are expressed in reciprocal degrees. The reason for the apparent inconsistency in nondimensionalization is historical and follows existing conventions.

The stability and control derivatives can be divided into longitudinal and lateral-directional derivatives, though there is some overlap in the control derivatives. Tables 1 and 2 tabulate the longitudinal and the lateral-directional stability and control derivatives, respectively. (In the tables and text, mathematical variables in parentheses follow the corresponding program variable names.)

TABLE 1 — LONGITUDINAL STABILITY AND CONTROL DERIVATIVES

	Force or moment		
State or control	Axial force	Normal force	Pitching moment
Aerodynamic bias	ca0 (C_{A_0})	cNorm0 (C_{N_0})	cm0 (C_{m_0})
Angle of attack	caa (C_{A_α})	cNorma (C_{N_α})	cma (C_{m_α})
Pitch rate	caq (C_{A_q})	cNormq (C_{N_q})	cmq (C_{m_q})
Elevator deflection	cade ($C_{A_{\delta_e}}$)	cNormde ($C_{N_{\delta_e}}$)	cmde ($C_{m_{\delta_e}}$)
d1 deflection	cad1 ($C_{A_{\delta_1}}$)	cNormd1 ($C_{N_{\delta_1}}$)	cmd1 ($C_{m_{\delta_1}}$)
d2 deflection	cad2 ($C_{A_{\delta_2}}$)	cNormd2 ($C_{N_{\delta_2}}$)	cmd2 ($C_{m_{\delta_2}}$)
d3 deflection	cad3 ($C_{A_{\delta_3}}$)	cNormd3 ($C_{N_{\delta_3}}$)	cmd3 ($C_{m_{\delta_3}}$)

TABLE 2 — LATERAL-DIRECTIONAL STABILITY AND CONTROL DERIVATIVES

	Force or moment		
State or control	Lateral force	Rolling moment	Yawing moment
Aerodynamic bias	cy0 (C_{Y_0})	cl0 (C_{ℓ_0})	cn0 (C_{n_0})
Angle of sideslip	cyb (C_{Y_β})	clb (C_{ℓ_β})	cnb (C_{n_β})
Roll rate	cyp (C_{Y_p})	clp (C_{ℓ_p})	cnp (C_{n_p})
Yaw rate	cyr (C_{Y_r})	clr (C_{ℓ_r})	cnr (C_{n_r})
Aileron deflection	cyda ($C_{Y_{\delta_a}}$)	clda ($C_{\ell_{\delta_a}}$)	cnda ($C_{n_{\delta_a}}$)
Rudder deflection	cydr ($C_{Y_{\delta_r}}$)	cldr ($C_{\ell_{\delta_r}}$)	cndr ($C_{n_{\delta_r}}$)
d3 deflection	cyd3 ($C_{Y_{\delta_3}}$)	cld3 ($C_{\ell_{\delta_3}}$)	cnd3 ($C_{n_{\delta_3}}$)
d4 deflection	cyd4 ($C_{Y_{\delta_4}}$)	cld4 ($C_{\ell_{\delta_4}}$)	cnd4 ($C_{n_{\delta_4}}$)

6.2.1.2 State initial conditions: Each state variable has a corresponding state initial condition increment parameter. The initial condition increment is added to each measured initial condition. The dimensions of each initial condition parameter are the same as those of its corresponding state variable. State initial condition parameters are v0 (V_0), alpha0 (α_0), q0 (q_0), theta0 (θ_0), beta0 (β_0), p0 (p_0), r0 (r_0), and phi0 (ϕ_0).

6.2.1.3 Instrumentation parameters: Numerous parameters characterize instrumentation installation and calibration. Each response variable has a corresponding response bias parameter. The bias value is added to the computed response variable value at each time point. The dimensions of each response bias parameter are the same as the dimensions of the corresponding response variable. Response bias parameters are vBias (V_b), alphaBias (α_b), qBias (q_b), thetaBias (θ_b), anBias (a_{n_b}), axBias (a_{x_b}), qdotBias ($\dot{q}_b$), betaBias (β_b), pBias (p_b), rBias (r_b), phiBias (ϕ_b), ayBias (a_{y_b}), pdotBias ($\dot{p}_b$), and rdotBias ($\dot{r}_b$).

Several response-measuring instruments have parameters describing their location in the aircraft. Each instrument-position parameter specifies the location of a response-measuring instrument aft of, to the right of, or above the aircraft reference point. Instrument-position parameters correct computed response values for instruments not located at the aircraft center of gravity. In operation, the program determines the distance from the instrument to the center of gravity by subtracting the instrument position from the center-of-gravity position (see section 6.2.2.1). The dimensions of all instrument-position parameters are feet. Table 3 gives the instrument-position parameter names.

TABLE 3 — INSTRUMENT POSITION PARAMETERS

	Aircraft axis		
Response variable	X	Y	Z
Velocity	xv (x_V)	yv (y_V)	zv (z_V)
Angle of attack	xa (x_α)	ya (y_α)	za (z_α)
Angle of sideslip	xb (x_β)	yb (y_β)	zb (z_β)
Axial acceleration	xax (x_{a_x})	yax (y_{a_x})	zax (z_{a_x})
Lateral acceleration	xay (x_{a_y})	yay (y_{a_y})	zay (z_{a_y})
Normal acceleration	xan (x_{a_n})	yan (y_{a_n})	zan (z_{a_n})

Two parameters define the flow amplification factors for the aircraft flow angle sensors. The upwash factor for the angle-of-attack sensor is ka (k_α), and the sidewash factor for the angle-of-sideslip sensor is kb (k_β). Both parameters are dimensionless.

6.2.1.4 Feedback gains: Five parameters define feedback gains for five of the state equations. Valid values for feedback gains are between 0 and 1; a value of 0 implies no feedback. While the feedback gains are parameters and can therefore be estimated, such use is experimental. Feedback gain parameters are gAlpha (g_α), gBeta (g_β), gP (g_p), gQ (g_q), and gR (g_r).

6.2.2 Constants.—The standard user routines define several constants. The use and dimensions of each constant are defined as follows.

6.2.2.1 Aircraft physical characteristics: Constants defining the mass characteristics of the aircraft are mass (m), ix (I_x), iy (I_y), iz (I_z), ixy (I_{xy}), ixz (I_{xz}), and iyz (I_{yz}). The mass is in slugs and all inertias in slug-feet squared. Constants defining the reference dimensions of the aircraft are area (s), span (b), and chord (c); the area is in feet squared and the span and chord are in feet. Constants

defining the location of the aircraft center of gravity are xcg (x_{cg}), ycg (y_{cg}), and zcg (z_{cg}); they are measured in feet aft of, to the right of, and above the aircraft reference point, respectively.

6.2.2.2 Time history variable averages: Several constants specify average values of measured time history variables. The avg_v, avg_alpha, avg_beta, avg_theta, and avg_phi constants contain average values for the corresponding response variables. The avg_qbar constant contains the average value of the dynamic pressure. Whenever a measured time history file is read (see section 4.2.1), the average value of each time history variable is computed, and values are assigned to the corresponding constants. Of course, at any point in the program, you can modify the value of a constant (see section 4.6.2). The equations of motion use the value of the constant variable containing the time history average of a response variable if its corresponding flag is turned on (see section 6.2.7).

6.2.3 States.—The standard user routines define eight state variables. All state variables are used internally in the program in English units. The wind-relative velocity is defined in spherical coordinates by three states: airspeed, angle of attack, and angle of sideslip. Airspeed is measured in feet per second, and its state name is v (V). Angles of attack and sideslip are both measured in degrees and are named alpha (α) and beta (β), respectively. The aircraft rotational velocities are defined in the aircraft body axes and are all measured in degrees per second. The roll, pitch, and yaw rate states are named p (p), q (q), and r (r), respectively. The aircraft attitude is defined by the the aircraft Euler angles, measured in degrees. The pitch and bank angles are named theta (θ) and phi (ϕ), respectively. Heading angle is not used in the aircraft equations of motion.

6.2.4 Controls.—The standard user routines define seven control variables. All control variables are measured in degrees. The three conventional aircraft control surface deflections, elevator, aileron, and rudder deflection, are named de (δ_e), da (δ_a), and dr (δ_r). The program defines four additional controls, d1 (δ_1), d2 (δ_2), d3 (δ_3), and d4 (δ_4).

6.2.5 Responses.—The standard user routines define 14 response variables. Each of the eight state variables (see section 6.2.3) has a corresponding response variable; the name and dimensions of each response variable are identical. Six additional response variables have no corresponding states. The aircraft rotational accelerations are defined in the aircraft body axes and are measured in degrees per second squared. The roll, pitch, and yaw accelerations are named pdot ($\dot{p}$), qdot ($\dot{q}$), and rdot ($\dot{r}$), respectively. The aircraft linear accelerations are defined in the aircraft body axes and are measured in g. The axial, lateral, and longitudinal accelerations are named ax (a_x), ay (a_y), and an (a_n), respectively.

6.2.6 Extras.—The standard user routines define three extra signals: Mach number, dynamic pressure, and altitude, named mach (M), qbar ($\bar{q}$), and alt (h), respectively.

6.2.7 Flags.—The standard user routines define seven flag variables. The flags select alternative forms of the equations or sources of data. Each flag specifies using the average value of a measured time history variable in the equations of motions in place of a time-varying quantity. This option is particularly useful when you do not have a measurement time history for a particular variable. In such a case, you can input an average value using the appropriate constant (see section 6.2.2) and then

activate the flag to use the average value in the equations. The flags are named use_avg_v, use_avg_beta, use_avg_theta, use_avg_phi, use_avg_mach, and use_avg_qbar.

National Aeronautics and Space Administration
Ames Research Center
Dryden Flight Research Facility
Edwards, California, September 9, 1986

APPENDIX A—PROGRAM STATUS FILE FORMAT

The program status file is a FORTRAN formatted file that stores the status of every program variable. Each record on the status file corresponds to an individual program variable or option; the records are grouped together into common record types. When the file is written by the program, a complete file is written; all records defined for the file are written. If you create the file outside the pEst program, it is not necessary to fill in all fields on the file. At startup, the program assigns a default value to each program variable; the value is retained unless it is redefined by reading a corresponding record on the status file. Except for the first record, which specifies the file format, the order of the records is immaterial. Any record can be repeated; if repeated, the last record read overrides all previous records. All fields are case insensitive; upper and lower case can be mixed.

Each record type is defined by its key word; the key word is the first field of each record and is left-justified in the first eight columns of the record. Each key word can be followed by several fields; the content and format of the fields, of course, depend on the record type. However, all fields of a common type have identical format, independent of the record type. There are no spaces between fields. Any field specifying a system or program variable name is 16 characters wide; the name is left-justified in the field. Any floating-point field is also 16 characters wide and is read with a FORTRAN g16.10 format. Any logical field is 2 characters wide, and contains either a t or an f, right-justified. All other fields are record dependent and are described in the individual sections that follow. All key words and literal phrases used in the status file are italicized in the following sections to distinguish them from other text.

A.1 Version Record

The file has a single version record. The version record specifies the format used when reading the file, hence the version record must be the first record on the file. The key word for the version record is *version*. Four fields follow: The first is a character string with the phrase *pest-current* left-justified in a field of 16 characters. The program recognizes the file by this phrase, and it is required. The second field specifies the version number of the current file as a character string, left-justified in a field 8 characters wide. The current value is 2.1. The third and fourth fields specify the date and time when the file is written by the program; these fields are not read by the program.

An example of a version record is

```
version pest-current     2.1      31-Mar-86 15:33:30
```

A.2 Title Record

The file typically has a single title record. The key word for the title record is *title*. A character field 40 characters wide specifying the title follows the key word.

An example of a title record is

```
title  F18  FLIGHT 11  LONG MAN 3
```

A.3 Parameter Record

The file can have several parameter records; typically there is one for each parameter defined in the program. The key word for a parameter record is *param*. Five fields follow: The first is the parameter name. The second and third are floating-point fields specifying the predicted and current values of the parameter, respectively. The fourth is a logical field specifying the estimation status of the parameter. The last is a floating-point field specifying the Cramér-Rao bound for the parameter.

Examples of parameter records are

```
param      cNormde          .7798399770E-02 .8341009928E-02 T .0001045977
param      clp             -.3680106990    -.3680106990     F .0000000000
```

A.4 Constant Record

The file can have several constant records; typically there is one for each constant defined in the program. The key word for a constant record is *const*. Two fields follow: The first is the constant name. The second is a floating-point field specifying the value of the constant.

Examples of constant records are

```
const   avg_alpha        8.165277248
const   ixz              1870.199950
const   zcg              5.317500110
```

A.5 Flag Record

The file can have several flag records; typically there is one for each flag defined in the program. The key word for a flag record is *flag*. Two fields follow: The first is the flag name. The second is the value of the flag, in a logical field.

An example of a flag record is

```
flag    use_avg_alpha    F
```

A.6 State Record

The file can have several state records; typically there is one for each state variable defined in the program. The key word for a state record is *state*. Three fields follow: The first is the state name. The second is a logical field specifying the status of the state equation. The third is a floating-point field specifying the integration limit for the state.

Examples of state records are

```
state   alpha            t 100000.0000
state   p                F 10000.00000
```

A.7 Response Record

The file can have several response records; typically there is one for each response variable defined in the program. The key word for a response record is *output*. Four fields follow: The first is the response name. The second is a floating-point field specifying the average value of the measured response variable. The third is a logical field specifying the status of the response equation. The last is a floating-point field specifying the response weighting in the cost function.

Examples of response records are

```
output  alpha          8.165277248     T 500.0000000
output  phi           -.3456887850     f 10.00000000
```

A.8 Control Record

The file can have several control records; typically there is one for each control variable defined in the program. The key word for a control record is *input*. Two fields follow: The first is the control name. The second is a floating-point field specifying the average value of the measured control variable.

Examples of control records are

```
input   de            -2.275935836
input   d3             .0000000000
```

A.9 Extra Record

The file can have several extra records; typically there is one for each extra variable defined in the program. The key word for a extra record is *extra*. Two fields follow: The first is the name of the extra signal. The second is a floating-point field specifying the average value of the measured extra variable.

An example of an extra record is

```
extra   qbar           112.4635873
```

A.10 Maneuver and Window Records

The file can have several maneuver or window records, or both; typically there is one for each maneuver or window defined in the program. The formats of the two records are identical; the only difference is the key word. The key word for a maneuver record is *maneuver*, while that for window record is *window*. Two fields specifying the start and end times follow the key word. The format for each time is identical. There are two blank spaces, followed by a two-digit hours specifier, a period, a two-digit minutes specifier, a period, a two-digit seconds specifier, a period, and a three-digit milliseconds specifier.

Examples of maneuver and window records are

```
maneuver  10.42.18.508  10.42.25.487
window    10.42.18.508  10.42.25.487
```

A.11 Option Records

The file can have several option records; typically there is one for each program option defined. The key word for an option record is *option*. Two fields follow: The first is the name of the option. There are six program options, each corresponding to a program variable. The second field is the value of the program variable, so the format depends on the variable. For the integMeth, minMeth, and gradMeth variables, the second field is a character string, left-justified in a field 16 characters wide. For the gradDelt and bound variables, the second field is a floating-point field. For the msgLevel variable, the second field is is an integer value right-justified in a field 5 characters wide.

Examples of option records are

```
option  integ          runge-kutta
option  min            newton-raphson
option  gradMeth       single-sided
option  gradDelt        .1000000000E-06
option  bound           .1000000000E-03
option  msgLevel          50
```

APPENDIX B—HELP FILES

The online help files used by pEst are listed in the following sections as they appear in program use. The help files detail the exact syntax of each command. The help files are alphabetically ordered.

B.1 Abort

```
abort [cmd] -- exit pEst immediately

USAGE
  abort

DESCRIPTION
  Exits pEst without saving current program status or writing
  computed time history.

EXAMPLES
  abort

SEE ALSO
  quit, save, write

KEYWORDS
  abort command,
  abort/exit pEst

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 11/19/85
```

B.2 Bound

```
bound [var] --  convergence bound

USAGE
  show bound
  set bound <convergence bound>

DESCRIPTION
  Bound is the value used in the convergence test of the
  minimization algorithm.  If the percentage change in the cost
  function between two successive iterations is less than this
  number, then convergence is declared and the program stops
  iterating.

  Convergence tests may become more complicated in future
```

```
  versions, possibly rendering this variable obsolete.

  The default is bound=.0001

EXAMPLES
  show bound
  set bound 0.00001

SEE ALSO
  show, set

KEYWORDS
  bound variable,
  convergence bound/criterion

AUTHOR James Murray
VERSION 2.1
DATE 11/21/85
```

B.3 Constant

```
const [cmd] -- display or set constants

USAGE
  const <const_list> <value>

PARAMETERS
  const_list
    List of const names, separated by commas or blanks.  May
    alternatively be 'all' (for all constants).  If not
    specified, it defaults to all constants.

  value
    Floating point value for all referenced constants.  If this
    value is present, all referenced constants will be set to
    the specified value.  If value is omitted, the constant
    values will remain unchanged.

DESCRIPTION
  Displays or sets value of selected constants.

EXAMPLES
  const area 200.0
  const all

SEE ALSO
  param, flag [cmd]
```

```
  constants [var]

KEYWORDS
  const command,
  set/change/list/show/display constants

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 3/11/86
```

B.4 Constants

```
constants [var] -- user-defined constants

DESCRIPTION
  The following are the names and brief descriptions of the user
  routine constants currently defined in pEst.  C.G positions
  are positive aft, above, and right of the reference point.
```

NAME	DESCRIPTION
avg_qbar	average qbar, psf
avg_mach	average mach
avg_v	average velocity, ft/sec
avg_alpha	average alpha, deg
avg_theta	average pitch attitude, deg
avg_beta	average beta , deg
avg_phi	average roll attitude , deg
mass	mass, slugs
ix	ix, slug-ft**2
iy	iy, slug-ft**2
iz	iz, slug-ft**2
ixy	ixy, slug-ft**2
ixz	ixz, slug-ft**2
iyz	iyz, slug-ft**2
area	reference area, ft**2
span	reference span, ft
chord	reference chord, ft
xcg	x-coordinate of cg, ft
ycg	y-coordinate of cg, ft
zcg	z-coordinate of cg, ft

```
SEE ALSO
  const, parameters, flags, states, responses, controls, extras

KEYWORDS
```

```
  constant names/descriptions

AUTHOR James Murray
VERSION 2.1
DATE 11/22/85
```

B.5 Controls

```
controls [var] -- control signals

DESCRIPTION
  The following are the names and brief descriptions of the
  control signals currently defined in pEst.

    NAME            DESCRIPTION

    de              elevator deflection, deg
    d1              deflection of control surface 1, deg
    d2              deflection of control surface 2, deg
    da              aileron deflection, deg
    dr              rudder deflection, deg
    d3              deflection of control surface 3, deg
    d4              deflection of control surface 4, deg

SEE ALSO
  parameters, constants, flags, states, responses, extras

KEYWORDS
  control/input signals/names/descriptions

AUTHOR James Murray
VERSION 2.1
DATE 11/22/85
```

B.6 Extras

```
extras [var] -- extra signals

DESCRIPTION
  The following are the names and brief descriptions of the
  extra signals currently defined in pEst.

    NAME            DESCRIPTION

    qbar            dynamic pressure, psf
    mach            mach number
```

```
    alt             altitude, ft

SEE ALSO
  parameters, constants, flags, states, responses, controls

KEYWORDS
  extra signals/names/descriptions

AUTHOR James Murray
VERSION 2.1
DATE 11/22/85
```

B.7 Flag

```
flag [cmd] -- display or set user-defined flags

USAGE
  flag <flag_list>  +active

PARAMETERS
  flag_list
    List of flag names, separated by commas or blanks.  May
    alternatively be 'all' (for all flags), 'active' (for the
    active flags), or one of several synonyms for active.  If
    not specified, it defaults to all active flags.

  +active (+on,+yes,+vary,+enable,+t,-inact,-deact,-disable,
    -no,-f)
    Switch to set the value of the referenced flags to active or
    inactive (or on/off, etc.) If this switch is present, all
    referenced flags will be set to the specified value.  If the
    switch is omitted, the flag values remain unchanged.

DESCRIPTION
  Displays or sets status of flags.

  The flag values are boolean (true/false, etc.) There is no
  distinction between +on, +true, +active, etc., except that
  some of these synonyms may sound more sensible than others for
  a specific flag.  The interpretation of the flag values is
  solely a function of the user routines.  The core pEst program
  does nothing with the flags except handle the mechanics of
  setting, displaying and storing them.

EXAMPLES
  flag use_avg_qBar,use_avg_mach +no
  flag all
```

```
SEE ALSO
  param, const [cmd]
  flags [var]

KEYWORDS
  flag command,
  set/change/list/show/display flags

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 3/18/86
```

B.8 Flags

```
flags [var] -- user-defined flags

DESCRIPTION

  The following are the names and brief descriptions of the user
  routine flags currently defined in pEst.


    NAME           DESCRIPTION

    use_avg_qbar   use average qbar in if true
    use_avg_mach   use average mach if true
    use_avg_v      use average velocity if true
    use_avg_alpha  use average alpha if true
    use_avg_theta  use average pitch attitude if true
    use_avg_beta   use average beta if true
    use_avg_phi    use average roll attitude if true

SEE ALSO
  flag, parameters, constants, states, responses, controls,
  extras, state

KEYWORDS
  flag names/descriptions

AUTHOR James Murray
VERSION 2.1
DATE 11/22/85
```

B.9 GradDelta

```
gradDelta [var] -- parameter increment to use for computing
  gradients

USAGE
  show gradDelta
  set gradDelta <value>

DESCRIPTION
  GradDelta is the relative step size used for computing finite
  difference gradients.  The step size used for each unknown
  parameter is gradDelta times the current estimate of the
  parameter.

  To avoid problems near zero, there is a fixed minimum
  parameter value in the step size computation.  If a parameter
  estimate is smaller than this limit (currently .000001), the
  step size for that parameter is gradDelta times the limit.

  The default is gradDelta=.0000001

EXAMPLES
  show gradDelta
  set gradDelta 0.00001

SEE ALSO
  show, set, gradMeth

KEYWORDS
  gradDelta variable,
  delta/increment/(step size) for computing gradients,
  sensitivity matrix

AUTHOR James Murray
VERSION 2.1
DATE 11/21/85
```

B.10 GradMeth

```
gradMeth [var] -- method of computing gradients

USAGE
  show gradMeth
  set gradMeth <method>
```

```
DESCRIPTION
  GradMeth is the method used by the program for computing
  gradients.  Two options are available.  Both are finite
  difference methods.  Pest has no provision for analytical
  differentiation.

  If gradMeth is set to '1' or 's[ingle-sided]', a forward
  difference method will be used.  This is the fastest method,
  requiring only n+1 integrations to compute an n-dimensional
  gradient.  It sacrifices some accuracy, however.

  If gradMeth is set to '2' or 'd[ouble-sided]', a central
  difference method will be used.  This is the most accurate
  method implemented, but requires 2n integrations to compute an
  n-dimensional gradient.  Since the integrations in the
  gradient computation dominate the computational time of pEst,
  the central difference method takes about twice the time of
  the forward difference method.

  For most efficient use of computer time (when it is worth the
  trouble), you might want to use forward difference computation
  most of the time, switching to central difference for the last
  few critical iterations.

  The default is gradMeth='single-sided'

EXAMPLES
  show gradMeth
  set gradMeth single

SEE ALSO
  show, set, gradDelta

KEYWORDS
  gradMeth variable,
  forward/central difference method,
  gradient/sensitivity computation

AUTHOR James Murray
VERSION 2.1
DATE 11/21/85
```

B.11 IntegMeth

```
integMeth [var] -- integration method

USAGE
```

```
   show integMeth
   set integMeth <method>

 DESCRIPTION
   IntegMeth is the method used by the program for integrating
   the equations of motion.  There are currently two valid
   values.

   If integMeth is E[uler], a first order Euler method is used.
   This simple method uses only 1 state derivative evaluation and
   1 response evaluation per time point.  Although fast, this
   method is quite innaccurate.  It is recommended only for rough
   approximations when computer time is of critical importance.

   If integMeth is R[unge-Kutta] a 4th order Runge-Kutta method
   is used.  This method requires 4 state derivative evaluations
   and 1 response evaluation per time point.  Although moderately
   expensive, its accuracy is good and it is recommended for most
   applications.

   The default is integMeth='Runge-Kutta'

 EXAMPLES
   show integMeth
   set integMeth runge

 SEE ALSO
   show, set

 KEYWORDS
   integMeth variable,
   Euler integration method/algorithm,
   Runge-Kutta/(Runge Kutta)/RK integration method/algorithm

AUTHOR James Murray
VERSION 2.1
DATE 11/21/85
```

B.12 Iterate

```
iterate [cmd] -- perform parameter estimation iterations

USAGE
  iterate [<niter>] [<min_meth>]

PARAMETERS
  niter
```

```
    Maximum number of iterations to do; if omitted or zero, the
    equations of motion will only be integrated and the cost
    function evaluated; no parameter update will be performed.

  min_meth
    Optional specification of the minimization method to be
    used.  If this parameter is included, the minMeth variable
    is set accordingly and the specified method is used.  If
    this parameter is not specified the current value of the
    minMeth variable is used.  The minMeth variable can also be
    set with the set command.  The 3 currently valid values are
    n[ewton-raphson] g[radient] and d[avidon-fletcher-powell].
    The initial default for min_meth is newton-raphson.  See the
    minMeth helpFile for details.

DESCRIPTION
  The iterate command initiates the parameter estimation.
  Anything beginning with 'it' will be accepted as an
  abbreviation.

EXAMPLES
  iterate
  iter 3 grad

SEE ALSO
  set, show, minMeth, integMeth, gradMeth, bound, msgLevel,
  param, state, response, flag

KEYWORDS
  iterate/iter/it command,
  set minMeth/(minimization method) variable,
  estimate parameters, integrate equations, minimize cost
  function

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 11/21/85
```

B.13 MinMeth

```
minMeth [var] -- cost function minimization method

USAGE
  show minMeth
  set minMeth <minimization method>

DESCRIPTION
```

MinMeth is the algorithm used by the program for minimizing the cost function. There are currently three valid values: n[ewton-raphson], g[radient], and d[avidon-fletcher-powell].

Newton-Raphson is recommended for most situations. It is actually implemented as a Gauss-Newton algorithm. Cramer-Rao bounds are available only when Newton-Raphson is used. The algorithm is modified by the addition of a line search in the Gauss-Newton direction. With this line search, the cost function is guaranteed never to increase from one iteration to the next; it may not decrease much (or at all), but it will never increase. Any violations of this principle should be reported as program bugs.

Gradient (also known as steepest descent) is in general less efficient than Newton-Raphson. The gradient algorithm is the most conservative of those implemented in that it has the least chance of failing. This is because the gradient method involves no matrix inversions. Note, however, that matrix inversion difficulties are usually a sign of more serious intrinsic identifiability problems and should not be lightly ignored. Like the pEst implementation of Gauss-Newton, the gradient method should never give a cost increase from one iteration to the next. Convergence of the gradient method is often extremely slow.

The implementation of the Davidon-Fletcher-Powell algorithm in the pEst program is still somewhat experimental. The algorithm is generally considered to be quite "powerful." However, we have not yet spent much time investigating its performance in the pEst environment.

The default is minMeth='newton-raphson'

EXAMPLES

```
show min_meth
set min_meth newton
```

SEE ALSO

show, set, iterate

KEYWORDS

minMeth variable,
Newton-Raphson/Newton/(Gauss-Newton)/Gauss method/algorithm,
gradient/(steepest descent) method/algorithm,
Davidon-Fletcher-Powell/dfp method algorithm,
cost function minimization/min/optimization method/algorithm

AUTHOR James Murray
VERSION 2.1
DATE 11/21/85

B.14 MsgLevel

msgLevel [var] -- amount of screen output

USAGE
  show msgLevel
  set msgLevel <message_level>

DESCRIPTION
  MsgLevel is the amount of screen output displayed. The following describes the output added at each value of msgLevel. Higher values of msgLevel also include all of the output for the lower levels. The numeric values assigned leave lots of room for future finer control.

  LEVEL DESCRIPTION

  (levels 1-9 control output that occurs only on error termination)

    5 Show error specifics, detailing where errors occur.

  (levels 10-49 control output that occurs once per iteration)

    10 Show cost function value each iteration.

    15 Show cost per response signal each iteration.

    20 Show estimated parameter values each iteration.

    40 Summarize each iteration.

  (levels 50-99 mostly control output that can be many times per iteration)

    50 Summarize each line search.

    52 Summarize each stage of line search.

    55 Show all integration failures, recoverable or not. At lower message levels, some integration failures are silently recovered, for instance by trying smaller step sizes.

```
    57   Show correlation matrix every newton-raphson iteration.

    60   Detail each cost evaluation in line searches.  This
         tracks the convergence of the line search algorithm.
         Mostly used in development of line search algorithms.

    62   Detail integration failures.  Show details of why
         integration was terminated.

    65   Show second gradient matrices every newton-raphson
         iteration.  Show approximate inverse second gradient
         matrices every dfp iteration.

    70   Show all gradient vectors.

    75   Show cost function value whenever it is evaluated for
         any purpose.

    80   Show the cost per response signal whenever the cost is
         evaluated.

  (levels 100-? control output that occurs every time point of
    integration)
  (Note that the volume of output can be extremely large)

   100   Show the state vector every time point. (Unimplemented)

  The default is msgLevel=50

EXAMPLES
  show msgLevel
  set msgLevel 60

SEE ALSO
  set, show

KEYWORDS
  msgLevel variable,
  msg,
  level/amount of output messages

AUTHOR James Murray
VERSION 2.1
DATE 11/22/85
```

B.15 Parameter

```
param [cmd] -- display or set parameter values or status

USAGE
  param <param_list>  <value>
        +active       +reset
        +predicted    +crBound      +delta

PARAMETERS
  param_list
    List of parameter names, separated by commas or blanks.  May
    alternatively be 'all' (for all parameters), 'active' (for
    the active parameters), or one of several synonyms for
    active.  If not specified, it defaults to all active
    parameters.  In this context, an 'active' parameter is one
    being allowed to vary in the estimation (varying is one of
    the acceptable synonyms).

  value
    Floating point value for all referenced parameters.  If this
    value is present, all referenced parameters will be set to
    the specified value.  If value is omitted, the parameter
    values will remain unchanged.

  +active (+on,+yes,+vary,+enable,+t,-inact,-deact,-disable,
    -no,-f)
    Switch to activate or deactivate estimation of all
    referenced parameters.  If +active is set, the referenced
    parameters will be estimated.  If -active is set, they will
    not be estimated.  If neither is set, the status of the
    parameters will remain unchanged.

  +reset
    If +reset is specified, all parameters in the param_list
    will be reset to their predicted values.  Note that it does
    not make much sense to specify both a value and +reset; if
    you do so, the +reset specification will override without
    comment.  The switch may be abbreviated to +r.  The default
    is -reset.

  +predicted
    If the +predicted switch is specified, the predicted values
    of the parameters will be included in the display resulting
    from this command.  The switch may be abbreviated to +pr.
    The default is -predicted.

  +crBound
```

If the +crBound switch is specified, the Cramer-Rao bounds of the parameters will be included in the display resulting from this command. Anything beginning with +cr or +b is accepted as a synonym. The default is -crBound.

+delta
If the +delta switch is specified, the the parameter value changes in the previous iteration will be included in the display. Anything begining with +d will be accepted as a synonym (unless the second letter is i, which could cause confusion with +disable).

DESCRIPTION
Displays or sets value and estimation status of selected parameters. Any command starting with 'par' will be accepted as a synonym.

EXAMPLES
```
param cma,cmde,cmq +fix
param cnp -0.25
parm vary
par all
```

SEE ALSO
const, flag [cmd]
parameters [var]

KEYWORDS
param command,
set/change/list/show/display parameter/param/parm values/ status

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 3/18/86

B.16 Parameters

parameters [var] -- estimated parameters

DESCRIPTION
The following are the names and brief descriptions of the estimated parameters currently defined in pEst. Instrument positions are positive aft, above, and right of the reference point.

NAME	DESCRIPTION
cNorm0	normal force aerodynamic bias
cNorma	normal force due to alpha, per deg
cNormq	normal force due to pitch rate, per rad
cNormde	normal force due to elevator deflection, per deg
cNormd1	normal force due to d1 deflection, per deg
cNormd2	normal force due to d2 deflection, per deg
cNormd3	normal force due to d3 deflection, per deg
cm0	pitching moment aerodynamic bias
cma	pitching moment due to alpha, per deg
cmq	pitching moment due to pitch rate, per rad
cmde	pitching moment due to elevator deflection, per deg
cmd1	pitching moment due to d1 deflection, per deg
cmd2	pitching moment due to d2 deflection, per deg
cmd3	pitching moment due to d3 deflection, per deg
ca0	axial force aerodynamic bias
caa	axial force due to alpha, per deg
caq	axial force due to pitch rate, per rad
cade	axial force due to elevator deflection, per deg
cad1	axial force due to d1 deflection, per deg
cad2	axial force due to d2 defection, per deg
cad3	axial force due to d3 deflection, per deg
v0	increment to velocity initial condition, ft/sec
alpha0	increment to alpha initial condition, deg
q0	increment to pitch rate initial condition, deg/sec
theta0	increment to pitch attitude initial condition, deg
vBias	measurement bias on velocity, ft/sec
alphaBias	measurement bias on alpha, deg
qBias	measurement bias on pitch rate, deg/sec
thetaBias	measurement bias on pitch attitude, deg
anBias	measurement bias on normal acceleration, g
axBias	measurement bias on axial acceleration, g
qdotBias	measurement bias on pitch acceleration, deg/sec**2
ka	upwash factor for alpha sensor
xa	x-coordinate of alpha sensor, ft
ya	y-coordinate of alpha sensor, ft
za	z-coordinate of alpha sensor, ft
xan	x-coordinate of normal accelerometer, ft
yan	y-coordinate of normal accelerometer, ft
zan	z-coordinate of normal accelerometer, ft

xax	x-coordinate of axial accelerometer, ft
yax	z-coordinate of axial accelerometer, ft
zax	z-coordinate of axial accelerometer, ft
xv	x-coordinate of velocity sensor, ft
yv	y-coordinate of velocity sensor, ft
zv	z-coordinate of velocity sensor, ft
cy0	side force aerodynamic bias
cyb	side force due to beta, per deg
cyp	side force due to roll rate, per rad
cyr	side force due to yaw rate, per rad
cyda	side force due to aileron deflection, per deg
cydr	side force due to rudder deflection, per deg
cyd3	side force due to d3 deflection, per deg
cyd4	side force due to d4 deflection, per deg
cl0	rolling moment aerodynamic bias
clb	rolling moment due to beta, per deg
clp	rolling moment due to roll rate, per rad
clr	rolling moment due to yaw rate, per rad
clda	rolling moment due to aileron deflection, per deg
cldr	rolling moment due to rudder deflection, per deg
cld3	rolling moment due to d3 deflection, per deg
cld4	rolling moment due to d4 deflection, per deg
cn0	yawing moment aerodynamic bias
cnb	yawing moment due to beta , per deg
cnp	yawing moment due to roll rate, per rad
cnr	yawing moment due to yaw rate, per rad
cnda	yawing moment due to aileron deflection, per deg
cndr	yawing moment due to rudder deflection, per deg
cnd3	yawing moment due to d3 deflection, per deg
cnd4	yawing moment due to d4 deflection, per deg
beta0	increment to beta initial condition, deg
p0	increment to roll rate initial condition, deg/sec
r0	increment to yaw rate initial condition, deg/sec
phi0	increment to bank angle initial condition, deg
betaBias	measurement bias on beta, deg
pBias	measurement bias on roll rate, deg/sec
rBias	measurement bias on yaw rate, deg/sec
phiBias	measurement bias on bank angle, deg
ayBias	measurement bias on lateral accelerometer, g
pdotBias	measurement bias on roll acceleration, deg/sec**2
rdotBias	measurement bias on yaw acceleration,

```
                    deg/sec**2
    kb              sidewash factor for beta sensor
    xb              x-coordinate of beta sensor, ft
    yb              y-coordinate of beta sensor, ft
    zb              z-coordinate of beta sensor, ft
    xay             x-coordinate of lateral accelerometer, ft
    yay             y-coordinate of lateral accelerometer, ft
    zay             z-coordinate of lateral accelerometer, ft
    gAlpha          feedback gain for alpha state
    gQ              feedback gain for q state
    gBeta           feedback gain for beta state
    gP              feedback gain for p state
    gR              feedback gain for r state

SEE ALSO
  param, constants, flags, states, responses, controls, extras

KEYWORDS
  parameter/param names/descriptions

AUTHOR James Murray
VERSION 2.1
DATE 3/10/86
```

B.17 pEst

```
pEst [cmd] -- parameter estimation program.

USAGE
  [/user/murray/commands/]pEst

DESCRIPTION
  This program does parmater estimation.  The program is
  designed for interactive operation.

  The program resides in the directory /user/murray/pEst, with
  aliases in /user/murray/commands.  The usePest command
  facilitates access to pEst.

  There is a full internal help facility which covers the
  commands within pEst.

EXAMPLES
  /user/murray/commands/usePest
  pEst
    read measured
    restore current
```

```
    iter 3 newton
    plot
    quit

CAVEATS
  Current dimensions limit the program to 200 parameters and
  2000 time points.  The limits are all checked, so that
  accidentally exceeding them will not cause the program to
  crash.

ERROR HANDLING
  The program attempts to recover from all errors.  Such mundane
  errors as exceeding dimension limits, or giving a non-existent
  file name or signal name are all caught.  The program should
  not crash, regardless of what junk you feed it for commands.
  Infinite or NaN quantities in the data may crash it.  If you
  succeed in crashing the program in any other way, please let
  me know.

SEE ALSO
  bindPEst, thPlot, usePest, internal help

FILES
  current  current program status.
  measured  time history data file.
  computed  time history file of estimated variables.
  pEst_thplot.temp  scratch file for communicating with thplot

IMPLEMENTATION
  Fortran program.

  The time history data file interface routines are used to read
  the data files.  See the help topic fileInterface for
  discussions of the file interface subroutines.  You must write
  customized versions of these routines to use pEst on data
  files not supported by the standard ones.

KEYWORDS
  program pEst,
  parameter estimation,
  mmle

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 10/28/85
```

B.18 Plot

```
plot [cmd] -- plot fits of currently active responses, control,
  and extras

USAGE
  plot [+res[iduals]] [+u | +c[ontrols] | +i[nputs]] [+ex[tras]]

PARAMETERS
  +res
    If this flag is turned on, the fit residuals are also
    plotted.  Default is -res.

  +u
    If this flag is turned on, the contol signals are also
    plotted.  Default is -u.

  +ex
    If this flag is turned on, the extra signals are also
    plotted.  Default is -ex.

DESCRIPTION
  Enters thPlot program and automatically plots time history
  fits of all currently active response variables.  Fit
  residuals and other signals are selectively plotted.  Upon
  completion of the time history plots, control is returned to
  pEst.

  This command differs from the thplot command in that the
  thplot command expects you to interactively enter any commands
  to the thplot program.  That allows more flexibility in what
  is plotted, but requires more input from the user.  The plot
  command, in contrast neither expects nor allows interactive
  input during the plotting.

EXAMPLES
  plot
  plot +res
  plot +u +extra

CAVEATS
  As currently implemented, the plot command creates a file
  called pEst_thplot.temp, used for input to the thplot program.
  Any pre-existing user file of that name will be overwritten
  (and later deleted).

ERROR HANDLING
  Automatically uses the write command to create the computed
```

```
  time history file.  Therefore, all of the error handling
  discussed under that command applies.  If the write command
  fails to write a computed time history file, the plot command
  will terminate and return to the pEst command line without
  doing any plots.  If a bad flag is specified, the plot command
  will terminate and return to the pEst command line without
  doing any plots.

SEE ALSO
  thplot, write

KEYWORDS
  plot command,
  plot response fits, residuals, controls/inputs, and extras

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 11/19/85
```

B.19 Quit

```
quit [cmd] -- save current status and exit pEst

USAGE
  quit

DESCRIPTION
  Writes current program status and computed time history files,
  and exits pEst.  The effect is the same as doing a save
  command with no parameter, followed by an abort command.

EXAMPLES
  quit

SEE ALSO
  abort, write, save

KEYWORDS
  quit command,
  save current status and quit/exit pEst

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 11/19/85
```

B.20 Read

```
read [cmd] -- read measured time history file

USAGE
  read [fileName]

PARAMETERS
  filename
    filename (or pathname) of the file to be read.  If not
    specified, it defaults to the same name as last specified on
    a read command.  If no fileName has been specified by any
    read command in the current run, the name defaults to
    'measured'.

DESCRIPTION
  Reads a measured time history data file.  A measured time
  history file must be read before pEst can do anything of
  consequence.  On initial startup, the program attempts to read
  a time history file named 'measured'.  If this attempt fails
  for any reason, you will be unable to do much until you do a
  successful read command.  The read command can also be used,
  after completing analysis of one maneuver, to switch to a new
  data set for analysis.

  A few program variables are automatically reset whenever a
  read command is successfully executed.  These are variables
  that are closely associated with the individual maneuver and
  unlikely to be meaningfully carried over from one case to
  another.

  Primarily, all maneuver and window times are redefined.  The
  maneuver times are automatically determined by looking for
  time dropouts of greater than 1 second (exact number subject
  to future change).  The window times are set to equal the
  maneuver times by default.

  The avg_ constants are also reset to the average of the
  measured data for the maneuver.  It is possible to manually
  override these values if you want to (for instance if the
  measured values are wrong), but it is unlikely that you want
  such overrides to automatically carry over from one maneuver
  to another (in fact it could cause great confusion if you were
  not aware that such carryover was occuring).  Therefore,
  average values are reset each time a maneuver is read.

EXAMPLES
  read
```

```
  read measured.case13

ERROR HANDLING
  The program gives error messages and returns to the command
  line on encountering any error.  Previously read data may be
  overwritten by a failed attempt to read new data; in this
  case, the program will clear out all of the corrupted data and
  you will have to execute a new successful read command before
  doing much else constructive.

  All commands that use time history data will recognize and
  appropriately handle the case where no valid time history data
  is available.  (Appropriate handling usually means that the
  command will give an error message and refuse to execute).

SEE ALSO
  write, save, restore

KEYWORDS
  read command,
  read/load time history data file

AUTHOR James Murray
VERSION 2.1
DATE 11/19/85
```

B.21 Response

```
response/output [cmd] -- display or set response equation status

USAGE
  response/output <resp_list> +active weight=<weighting>

  resp_list
    List of response variable names, separated by commas or
    blanks.  May alternatively be 'all' (for all responses),
    'active' (for the active responses), or one of several
    synonyms for active.  If not specified, it defaults to all
    active responses.

  +active (+on,+yes,+vary,+enable,+t,-inact,-deact,-disable,
    -no,-f)
    Switch to activate or deactivate computation of the
    referenced responses.  If +active is set, the referenced
    response equation(s) will be activated.  If -active is set,
    they will be deactivated.  If neither is set, the status of
    the response equations will remain unchanged.
```

```
  weighting
    Floating point value for response error weighting.  If this
    parameter is specified, the cost function weightings of all
    the referenced outputs will be set to the specified value.
    If it is not specified, the weightings will be left
    unchanged.  The keyWord may be abbreviated to anything
    starting with w.

DESCRIPTION
  Displays or sets status of response variable(s).  Anything
  beginning with 'resp' or 'out' will be accepted for this
  command.

  Note that the weighting is irrelevant for responses that are
  inactive.  An inactive response makes no contribution to the
  cost function, making its effective weighting zero.  The
  differences between setting the weighting to 0 and
  deactivating the response are two: First, the responses are
  still computed (just not used in the cost function) if the
  weighting is zero for an active response; if the response is
  inactive, it is never computed.  Obviously, if you need the
  response computation for some reason other than its use in the
  cost function (perhaps you want a plot of it), you must have
  the response active.  This may involve substantial computation
  time, so if you don't need the response, it is preferrable to
  deactivate it.  The second difference is just one of
  convenience: if you deactivate a response, the program still
  remembers what its weighting was in case you later want to
  re-activate it with the same weighting as before.

EXAMPLES
  response
  resp alpha +off
  output beta +activate w=100.0

SEE ALSO
  state [cmd]
  responses [var]

KEYWORDS
  response command,
  set/list/show/display response/output equation/variable
    status/weighting

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 3/18/86
```

B.22 Responses

```
responses [var] -- response signals

DESCRIPTION
  The following are the names and brief descriptions of the
  response signals currently defined in pEst.

    NAME            DESCRIPTION

    v               velocity, ft/sec
    alpha           angle of attack, deg
    q               pitch rate, deg/sec
    theta           pitch attitude, deg
    an              normal acceleration, g's
    ax              longitudinal acceleration, g's
    qdot            pitch acceleration, deg/sec**2
    beta            angle of sideslip, deg
    p               roll rate, deg/sec
    r               yaw rate, deg/sec
    phi             bank angle, deg
    ay              lateral acceleration, g's
    pdot            roll acceleration, deg/sec**2
    rdot            yaw acceleration, deg/sec**2

SEE ALSO
  response, parameters, constants, flags, states, controls,
    extras

KEYWORDS
  response/output signals/names/descriptions

AUTHOR James Murray
VERSION 2.1
DATE 11/22/85
```

B.23 Restore

```
restore [cmd] -- restore status from file

USAGE
  restore [fileName]

PARAMETERS
  filename
    filename (or pathname) of the file to be read.  If not
    specified, it defaults to the same name as last specified on
```

a restore command. If no fileName has been specified by any restore command in the current run, the name defaults to 'current'.

DESCRIPTION

Reads program status from the specified file. The program status includes the values of all parameters and program options. The file can be from either of two sources. The file may have been created by an independent program, usually by looking up predicted derivative values from a data table; this is often the case when starting a new case. Alternatively, the file can have been created from a save command; in this case, the program will be restored to the same status as when the save command was executed.

A file created while analyzing one case can be restored while analyzing a different set of time history data. This allows, for instance, a convenient way of starting the estimation from converged values obtained from a case at a similar flight condition. Some values, such as the parameter estimates, can be meaningfully transfered from one case to another in this manner. However, other values on the current status file are unlikely to be useful (or legal) when applied to a different time history data file.

In particular, the window definitions are closely tied to specific maneuvers. Window times for one case are likely to be outside of the maneuver times for a different case.

The avg_ constants are also closely tied to the specific maneuvers; they usually equal the average measured values for the maneuver. It is possible to manually override these values if you want to (for instance if the measured values are wrong), but it is unlikely that you want such overrides to automatically carry over from one maneuver to another (in fact it could cause great confusion if you were not aware that such carryover was occuring).

To avoid these undesired carryovers, while allowing carryover of other pertinent parameters, the restore command checks for consistency between the status file and the time history data being analyzed. The status file includes a record of the maneuver times and average measured signal values to facilitate this comparison. If the data on the status file is consistent with the time history data being analyzed, then all parameters from the status file are accepted, including window times and avg_ constants. If the data on the status file is inconsistent with the maneuver being analyzed, then the window

times and avg_ constants on the status file are rejected; instead, the windows are set equal to the available maneuvers and the avg_ constants are set equal to the average measured values.

For similar reasons, the window times and avg_ constant values are reset every time a read command is executed.

On initial startup, if the automatic read command works, the program automatically attempts a restore command for the file name 'current'.

Unlike the read command, which must succeed before you can analyze any data, the restore command is technically optional. It is possible to analyze a case without ever reading a status file. All of the items set by the restore command have default values. In some cases where no meaningful default is possible, the default value is an illegal one (for instance weight defaults to 0). Such values must be reset either by a restore command or by manual entry before estimation can procede. The program will give an error message if you attempt estimation without setting these variables to legal values.

EXAMPLES

```
restore
restore current.case13
```

CAVEATS

Because of the interaction between the read and restore commands, you must be cautious of the order in which you execute them. If you want to do a read command and a restore command to completely restore a previously saved status, the read command must precede the restore command.

The data on the saved file is in ascii form, accurate to 'only' about 10 digits. For 'reasonable' situations, this should be far more accuracy than you will need. For highly unstable systems, it is possible for the rounding to 10 digits to make the restored state noticably different than that originally saved. The validity of the estimates for any case in which this effect is noticable is in serious doubt anyway.

SEE ALSO

save, read

KEYWORDS

restore command,

```
  restore/read/load current status

AUTHOR James Murray
VERSION 2.1
DATE 11/19/85
```

B.24 Save

```
save [cmd] -- save current status to a file.

USAGE
  save [fileName]

PARAMETERS
  filename
    filename (or pathname) of the file to be written.  If not
    specified, it defaults to the same name as last specified on
    a RESTORE command.  If no fileName has been specified by any
    restore command in the current run, the name defaults to
    'current'.

DESCRIPTION
  Saves current program status to a file.  The program status
  includes the values of all parameters and program options.
  The resulting file can be used for several purposes: The
  program can later be restored to the saved point by using the
  restore command.  The file can be input to plotting programs
  or other programs that use the parameter values estimated by
  pEst.  The file can also be printed (it is ascii) as a record
  of the results.

EXAMPLES
  save
  save current.case23.19Nov85

CAVEATS
  Note that the default fileName is taken from the last RESTORE
  command, rather than the last SAVE command.  The logic behind
  this is that a save command with no arguments is taken as a
  request to save the current status in place of the file that
  you started with; this is the most usual mode of operation.  A
  save specifying a different fileName is assumed to be asking
  to save a specific status to a specific file for special
  purposes.  All subsequent work reverts to the original file.

BUGS
  By normal Fortran defaults, the file is created with the
```

```
  +fortranCCTL flag set, in spite of the fact that the file does
  not have fortran carriage control characters in column 1.  In
  order to correctly print the file, you must explicitly specify
  -fortranCCTL in the print command (or do a modifyFile to
  correctly set the flag maintained with the file).

SEE ALSO
  restore, write

KEYWORDS
  save command,
  write/save current status

AUTHOR James Murray
VERSION 2.1
DATE 11/19/85
```

B.25 Set

```
set [cmd] -- set program variables.

USAGE
  set <variable-name> <value>

PARAMETERS
  variable-name
    Name of the variable to be set.

  value
    Value to be used for the variable.  The syntax and legal
    values may vary for different variables.  In particular,
    some variables have several components and the value syntax
    must specify which component is being set.

DESCRIPTION
  The set command sets the values of program variables.  For a
  list of the available variables, do "help variables".  For
  details about a specific variable, do help on the variable
  name.

EXAMPLES
  set msgLevel 80
  set integMeth euler
  set gradDelta 0.00001
  set window time 0.5 9.5

SEE ALSO
```

```
  show

KEYWORDS
  set command,
  set/change program variable values

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 11/20/85
```

B.26 Show

```
show/list/display [cmd] -- show values of program variables

USAGE
  show/list/display <variable-name>

DESCRIPTION
  Shows the value of the specified program variable.  For a list
  of the available variables, do "help variables".  For details
  about a specific variable, do help on the variable name.

  List and display are accepted as synonyms.  Sh and disp are
  acceptable abbreviations.

EXAMPLES
  list integ
  show bound
  display window
  show statistics

SEE ALSO
  set

KEYWORDS
  show/list/display command,
  show/list/display program variable values

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 11/20/85
```

B.27 State

```
state [cmd] -- display or set state equation status
```

USAGE

state <state_list> +active limit=<limit>

PARAMETERS

state_list

List of state names, separated by commas or blanks. May alternatively be 'all' (for all states), 'active' (for the active states), or one of several synonyms for active. If not specified, it defaults to all active states.

+active (+on,+yes,+vary,+enable,+t,-inact,-deact,-disable, -no,-f)

Switch to activate or deactivate integration of the referenced states. If +active is set, the referenced state equation(s) will be integrated. If -active is set, they will not be integrated. If neither is set, the status of the state equations will remain unchanged.

limit

The limit on the absolute value of the referenced states allowed during integration. Any integration that exceeds this limit will be abandoned. If this parameter is not specified, the previous limits remain unchanged.

DESCRIPTION

Displays or sets status of state equation(s).

The system equations can change fairly significantly depending on which states are active and which ones are inactive. If the state is inactive, no computed value for that state variable is defined; thus response equations or other state equations that use the deleted state variable must be revised. The revision done is to substitute some other quantity for the unavailable state variable.

By default, if a state is deactivated, the corresponding measured quantity is substituted. Any output bias on the measured quantity is subtracted before substituting it. Also the measured flow angles are corrected to the center of gravity before substituting them.

The variables with names like use_avg_alpha can force alternate substitutions. If use_avg_alpha is true, then the constant avg_alpha is substituted wherever the computed alpha state would otherwise have been used. In this case, the substitution occurs even if the alpha state is active. Corresponding comments apply to the other states.

```
EXAMPLES
  state theta +on limit=1000
  state all
  state phi -active

SEE ALSO
  response [cmd]
  states [var]

KEYWORDS
  state command,
  set/change/list/show/display state equation/variable status,
  add/delete state equations

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 3/17/86
```

B.28 States

```
states [var] -- state variables

DESCRIPTION
  The following are the names and brief descriptions of the
  state variables currently defined in pEst.

    NAME            DESCRIPTION

    v               velocity, ft/sec
    alpha           angle of attack, deg
    q               pitch rate, deg/sec
    theta           pitch attitude, deg
    beta            angle of sideslip, deg
    p               roll rate, deg/sec
    r               yaw rate, deg/sec
    phi             bank angle, deg

SEE ALSO
  state, parameters, constants, flags, responses, controls, extras

KEYWORDS
  state variables/names/descriptions

AUTHOR James Murray
VERSION 2.1
DATE 11/22/85
```

B.29 Statistics

```
stats [var] -- time history statistics

USAGE
  show stat[istics]

DESCRIPTION
  The 'show statistics' command lists the average value of every
  measured response, input, and extra signal for the time
  history data currently loaded.

  Statistics is not really a 'variable' as it can not be set; it
  can only be displayed.

EXAMPLES
  show stats
  show statistics

SEE ALSO
  show

KEYWORDS
  statistics variable,
  maneuver/signal statistics/stats/averages

AUTHOR James Murray
VERSION 2.1
DATE 11/21/85
```

B.30 thPlot

```
thplot [cmd] -- execute thPlot program

USAGE
  thplot

DESCRIPTION
  Writes time histories of computed responses, then enters
  thPlot program.  Upon termination of the thPlot program,
  control is returned to pEst.  The computed responses are
  written by an automatic write command with no arguments;
  therefore the file name is whatever such a write command would
  use (usually the name 'computed')

  This command differs from the plot command in that the thplot
  command expects you to interactively enter any commands to the
```

```
  thplot program.  This allows more flexibility in what is
  plotted, but requires more input from the user.  The plot
  command, in contrast neither expects nor allows interactive
  input during the plotting.

EXAMPLES
  thplot

ERROR HANDLING
  Automatically uses the write command to create the computed
  time history file.  Therefore, all of the error handling
  discussed under that command applies.  If the write command
  fails to write a computed time history file, the plot command
  will terminate and return to the pEst command line without
  doing any plots.

SEE ALSO
  plot, write

KEYWORDS
  thplot command,
  time history plots

AUTHOR James Murray - NASA Dryden
VERSION 2.1
DATE 11/19/85
```

B.31 Title

```
title [var] --  title for current file and plots.

USAGE
  show title
  set title '<title>'

DESCRIPTION
  Title is the title used on the current file and on the top of
  each plot page.  The length of the title is limited to 40
  characters; longer titles are truncated without comment.  If
  the title contains imbedded blanks (not uncommon), the title
  must be enclosed in quotes; otherwise the title will be
  truncated to the first word.

  The default is title = ' '.

EXAMPLES
  show title
```

```
  set title 'x29 flight 20 case 10'

SEE ALSO
  show, set

KEYWORDS
  title variable,
  plot title

AUTHOR James Murray
VERSION 2.1
DATE 2/19/86
```

B.32 Version

```
version [topic] -- changes in pEst with version 2.1

DESCRIPTION
  pEst version 2.1 is now released.  The following describes the
  changes between version 2.1 and the previous release, version
  2.0.  Access to the new version is automatic when you use pEst
  normally.  The previous version will be retained for an
  interim period as a backup.  To access the previous version,
  use the command pEst2.0.

COMPATABILITY
  Version 2.1 has several small differences from version 2.0.
  The exact syntax of a few of the commands has changed.  This
  will require getting used to the revised syntax and changing
  any command files; the changes are all small.

  If you have a customized set of user routines, you must make a
  minor change to the argument list of defineNames; this change
  is small and mostly involves deleting a few lines from the
  routine.

  There are several additions to the current file, but version
  any current files from version 1.6 or later will be accepted
  without error.  Any program that reads current files must be
  able to accept the new fields.

USER ROUTINES
  All the arguments have been removed from subroutine
  defineNames.  They were not being used for anything
  constructive anyway and they caused excessive clutter in the
  main program.
```

The semantics of the interface to subroutine computeZ have changed. We now allow computeZ to change its xc (computed state) argument. Any such change constitutes the correction step of a predictor-corrector algorithm such as a Kalman filter. This change is solely one of establishing an expanded convention. It requires no code changes in old code, but just liberalizes what new code is allowed to do.

USAGE

The param command has been expanded to allow optional display of or reset to the predicted values. See the param helpFile for details.

The title variable has been added to allow the title used on the current file to be modified. The same title is now also put on time history plots. See the title helpFile for details.

The plot command uses thPlot's +expand option to expand scales to the full screen width.

An extension to the state command allows the state variable limits to be displayed and changed. These limits are also now saved on the current file. The state variable limits are what stops the integration when it goes unstable.

There are 5 new parameters (gAlpha,gQ,gBeta,gP,gR) in the default user routines. These parameters are gains in the filter error formulation. Among other things, they can be used to stabilize the estimation for unstable systems. Their use at this time is still experimental.

DOCUMENTATION

The helpfiles mentioned in the following "SEE ALSO" section have been modified to reflect the changes. The helpFile format for showing command syntax has been redone to be more consistent with Elxsi helpFiles (also a little simpler). This file is included under "help version".

SEE ALSO

param,const,flag,state,response,

KEYWORDS

pEst version 2.1 changes

AUTHOR James Murray
VERSION 2.1
DATE 3/17/86

B.33 Window

```
window [var] --  maneuver window descriptor

USAGE
  show window
  set window [<window number>] [man[=]<maneuver number>]
    [time[=]<start> <end>]

DESCRIPTION
  A window is a time interval which is wholly contained within a
  maneuver time interval.  A maneuver may contain multiple
  windows.

  When the equations of motion are evaluated, the computed time
  history is generated only for those time points contained
  within the window(s).  Outside of the window(s), the average
  value for the output variables is used.  Also, the cost
  function for the estimation process depends only on those time
  points within the window(s).

  If, when a window is defined, the window number is not
  specified, all existing windows will be deleted and the new
  window will be window 1.  The window number, if specified,
  cannot exceed the current window count by more than 1.  If the
  maneuver number is not specified, the default maneuver number
  1 will be used.  The window start and end times must be
  specified and are given relative to the starting time of the
  maneuver.

  The default is that the windows exactly equal the available
  maneuvers.

EXAMPLES
  show windows
  set wind 1 time=0.5-9.5
  set wind man=3 time 1.25 5.55

SEE ALSO
  show, set

KEYWORDS
  window variable,
  multiple maneuvers,
  start/stop times,
  time segments/intervals/windows

AUTHOR James Murray
```

VERSION 2.1
DATE 11/21/85

B.34 Write

```
write [cmd] -- write computed time history file

USAGE
  write [fileName]

PARAMETERS
  filename
    filename (or pathname) of the file to be written.  If not
    specified, it defaults to the same name as last specified on
    a write command.  If no fileName has been specified by any
    write command in the current run, the name defaults to
    'computed'.

DESCRIPTION
  Writes a computed time history data file.  This file is most
  often used for plotting, but can have numerous applications.
  It could be input to programs to analyze the residual
  statistics.  It also allows pEst to be used as a simple batch
  simulator for other purposes.

  The signals written to the file are the calculated states
  (-s-hat suffixes), the calculated responses (-hat suffixes),
  and the residuals (-res suffixes).  The names of all signals
  are derived by appending the indicated suffixes to the basic
  state or response variable name.  The residual is defined to
  be the measured response minus the calculated response.

EXAMPLES
  write
  write computed.case12

ERROR HANDLING
  If the output file can not be opened or if integration of the
  equations is disallowed for any reason, the write command will
  give an error message and return to the command line.

  If the integration fails part way through (usually because it
  goes unstable), the file is written normally up to the point
  where the integration failed.  The remainder of the file is
  written with average measured values substituting for the
  unavailable calculated responses.  This is done to accomodate
  the plotting program, which wants the same time points in both
```

the measured and calculated data files in order to plot a fit.

SEE ALSO
read, plot, thplot

KEYWORDS
write/save command,
write/save computed time history data

AUTHOR James Murray
VERSION 2.1
DATE 11/19/85

REFERENCES

1. Taylor, Lawrence W., Jr.; and Iliff, Kenneth W.: A Modified Newton-Raphson Method for Determining Stability Derivatives From Flight Data. Second International Conference on Computing Methods in Optimization Problems, San Remo, Italy, Sept. 9-13, 1968, Academic Press, New York, 1969, pp. 353-364.
2. Maine, Richard E.; and Iliff, Kenneth W.: User's Manual for MMLE3, A General FORTRAN Program for Maximum Likelihood Parameter Estimation. NASA TP-1563, 1980.
3. Maine, Richard E.: Manual for GetData Version 3.1—A Fortran Utility Program for Time History Data. NASA TM-88288, 1987.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-88280		

4. Title and Subtitle	5. Report Date
pEst Version 2.1 User's Manual	September 1987
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
James E. Murray and Richard E. Maine	H-1390
	10. Work Unit No. RTOP 505-68-31
9. Performing Organization Name and Address NASA Ames Research Center Dryden Flight Research Facility P.O. Box 273, Edwards, CA 93523-5000	11. Contract or Grant No.
	13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This report is a user's manual for version 2.1 of pEst, a FORTRAN 77 computer program for interactive parameter estimation in nonlinear dynamic systems. The pEst program allows the user complete generality in defining the nonlinear equations of motion used in the analysis. The equations of motion are specified by a set of FORTRAN subroutines; a set of routines for a general aircraft model is supplied with the program and is described in the report. The report also briefly discusses the scope of the parameter estimation problem the program addresses. The report gives detailed explanations of the purpose and usage of all available program commands and a description of the computational algorithms used in the program.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Parameter estimation Stability and control derivatives	Unclassified — Unlimited Subject category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	75	A04

**For sale by the National Technical Information Service, Springfield, Virginia 22161.*